Modellbasierte Anforderungsspezifikation sicherheitskritischer Systeme im Bahnbereich – Beschreibungsmittel und ihre Anwendung

Von der
Fakultät Architektur, Bauingenieurwesen und Umweltwissenschaften
der Technischen Universität Carolo-Wilhelmina
zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)
genehmigte

Dissertation

von
Oliver Lemke
geboren am 03.02.1975
aus Frankfurt/M.

Eingereicht am: 01.07.2009
Disputation am: 18.05.2010

Berichterstatter: Univ.-Prof. Dr.-Ing. Jörn Pachl
PD Dr. Hardi Hungar

2010

Inhaltsverzeichnis

1. Abstract

In common with other technical sectors, control and safety functions in the rail sector are performed in numerous places by reactive systems. In many of these systems the actual functionality is implemented in software. Many systems also carry responsibility for human life and material assets.

The very important job of ensuring the correct functioning of the system begins with the creation of the requirements specification. This must contain all necessary information for a manufacturer to develop and produce the system and to maintain the system over its complete life-cycle. The requirements specification therefore is a key element in the complete life-cycle of a system.

Currently mainly informal description techniques are used for the specification of requirements in the rail sector. These techniques are, to a lesser or greater extent, open to interpretation and subject to ambiguities and misunderstandings. In addition they do not allow automatic checks for correctness and inconsistencies. This can lead to the introduction of functional errors already in the requirements definition phase of the system development process.

In the last few years several new ideas and concepts have been presented in various papers as to how the specification of systems can be improved. The research often focuses on the adaptation or extension of existing description instruments such as the Unified Modeling Language (UML) and mainly focuses on the system development phase of the manufacturer. Additionally the integration of description instruments, methods and tools to a stringent overall concept, necessary for their productive deployment, is not covered.

This research therefore develops an overall concept, which, for the first time, combines a semi-formal model-based description instrument with a process model for the creation of requirements specifications. Test and verification methods, required for safety-critical systems, are also an integral part of this concept.

The standardized Systems Modeling Language (SysML) from the Object Management Group (OMG) is used as the description instrument. First a subset of the language (SysML (A)) is identified, which appears to be particularly suitable for the description of requirements models in the rail sector. These language elements are then applied, within a fixed model architecture, to the description of several aspects of a future system, such as the interface with the environment, system functions and system behavior. Grouping and organization of the requirement model - absolutely necessary in practice - is realized through so-called “views” and a two-level recursive hierarchy of systems and subsystems. Interfaces are also provided in order to couple information with non-functional requirements, which cannot be sufficiently described in the modeling language. This part of the work is completed with a listing of consistency conditions, which the requirements model must meet in order that the integrity of the model content is guaranteed across all grouping levels.

The second significant component of the work is a process model, which defines the model creation procedure. As opposed to system development processes, which are carried out later in the life-cycle and build on the requirements specification, one is confronted with a minimal amount of mostly vague and quickly-changing information. The process model must allow for these constraints.

Therefore an iterative-incremental approach, grouped into phases and subprocesses, is used, with which the amount of concrete information grows step-by-step. Test and verification techniques are used in each step in order to guarantee the correctness of the modeling work previously carried out. The precondition for this is that a model is obtained as early possible, which is mature enough for model execution to take place. This allows the application of automated test and verification techniques and the simulation of the future behavior of the system. Within the scope of this research an automated test case generation tool is used which allows the system behavior to be made explicit such that it can be compared against the specified behavior. Optionally a formal verification tool (based on model-checking) can be integrated, such that the requirement model can be tested against safety constraints.

Through the course of this research a complete example model is used. This consists of a requirement model for a level crossing safety system, which is oriented to the functionality of a real safety system. The example model was developed to complete model execution capability and the test methods described above were applied to it.

This showed that:

- model-based requirement specification is possible
- the chosen process model can be applied in practice
- and through the strict process orientation and the use of contemporary technologies it is possible to improve the quality of requirement specifications.

The research also introduces the tools necessary for the individual steps and analyzes their applicability. This leads to statements about a prototype tool-chain, with which the procedure described can be implemented.

2. Einleitung

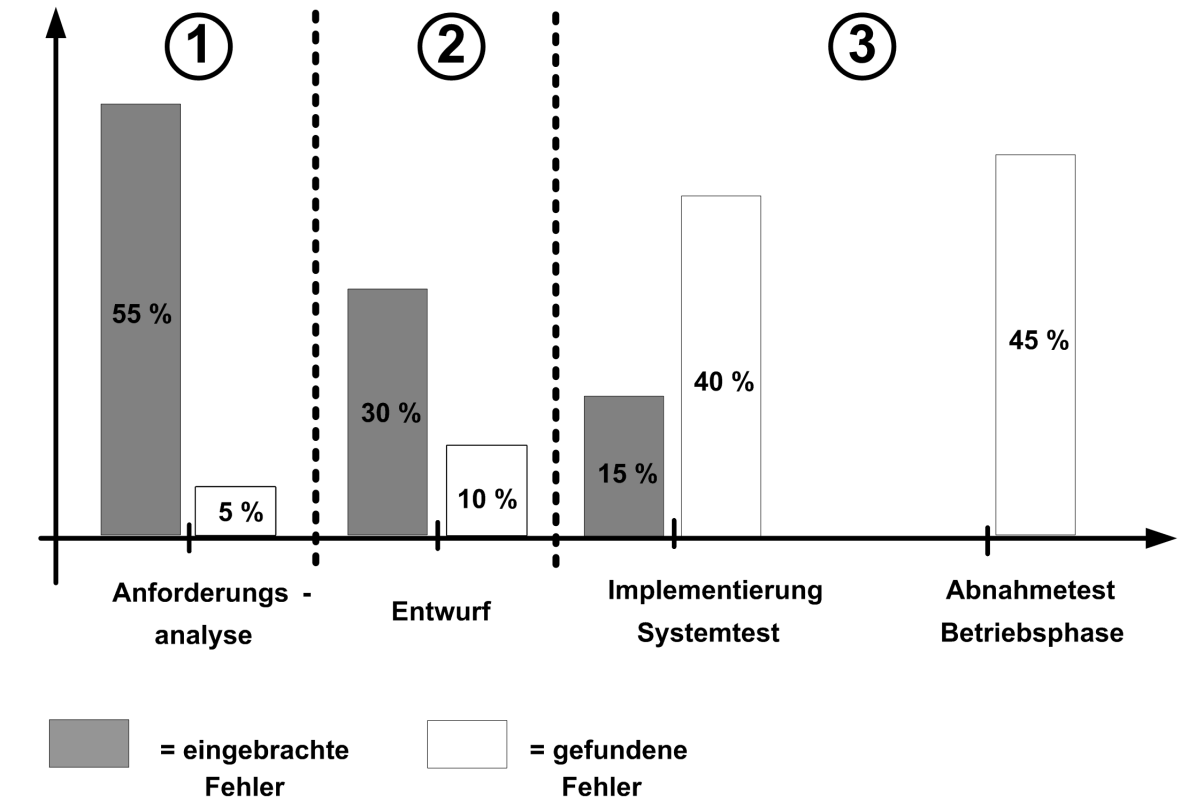
Wie in anderen technischen Bereichen auch, übernehmen im Eisenbahnwesen an zahlreichen Stellen reaktive Systeme unterschiedlicher Komplexität Steuerungs- und Sicherungsfunktionen. Das Spektrum reicht von Fahrgastinformationssystemen, über Tür-, Antriebs- und Bremssteuerungen, ETCS-Fahrzeuggeräten bis hin zu komplexen Systemen wie beispielsweise elektronischen Stellwerken. Bei vielen dieser Systeme wird die eigentliche Funktionalität durch Software realisiert, viele dieser Systeme übernehmen zudem Sicherheitsverantwortung für Menschenleben und Sachwerte. Diese software-lastigen, sicherheitsrelevanten Systeme sind Betrachtungsgegenstand der folgenden Ausführungen in dieser Arbeit.

Die korrekte Funktion dieser Systeme sicherzustellen, ist auf Grund der Sicherheitsrelevanz eine wichtige und durch Normen und Richtlinien stark reglementierte Aufgabe, die sich über den gesamten Lebenszyklus des Systems erstreckt. Am Anfang dieses Lebenszyklus steht die Erstellung einer Anforderungsspezifikation durch den Auftraggeber und späteren Betreiber des Systems. Diese muss alle Informationen enthalten, die ein Auftragnehmer benötigt, um das System zu entwickeln, zu produzieren und während der Betriebszeit zu begleiten. Die Anforderungsspezifikation ist außerdem Vertragsbestandteil: Sie legt den Umfang und die Funktionen des zu liefernden Systems fest und ist damit die Grundlage für die spätere Abnahme und letztendlich auch für die buchhalterische Abwicklung des Projekts. Zudem muss von der jeweiligen Aufsichtsbehörde - beispielsweise dem Eisenbahnbundesamt - eine Zusicherung auf Grundlage der Anforderungsspezifikation ausgesprochen werden, die die Konformität des zu entwickelnden Systems mit den entsprechenden Gesetzen, Normen und Richtlinien bestätigt. Die Anforderungsspezifikation besitzt damit für alle beteiligten Parteien große Bedeutung. Diese wichtige Funktion wird beispielsweise in [GEA08] folgendermaßen beschrieben:

> „Die Spezifikation der Anforderungen hat entscheidende Bedeutung, da sie über die Messbarkeit der Ergebnisse und damit über die erreichbare Qualität der RAMS-Aktivitäten entscheidet. [...] Ziel ist es, die Anforderungen an das System so genau wie möglich, dem jeweiligen Zweck des Projektes, seiner Komplexität und dem abgestimmten Zusammenwirken der Partner entsprechend zu definieren."

Wegen dieser herausgehobenen Bedeutung der Anforderungsspezifikation werden an deren Qualität und insbesondere an ihre inhaltliche Korrektheit besondere Ansprüche gestellt. Betrachtet man jedoch die „Chaos-Studie" der Standish Group [STA98], zeigt sich, dass dieses hohe Qualitätsziel oftmals nicht erreicht wird. In dieser Studie wurde untersucht, wann innerhalb des Lebenszyklus komplexer Softwareprojekte[1] Fehler eingeschleppt wurden und wann sich diese Fehler letztendlich offenbarten. Das obere Diagramm in Abbildung 2.1 zeigt die ermittelten Ergebnisse. Über die Hälfte der später aufgedeckten Fehler wurde dabei bereits in der Anforderungsspezifikation eingebracht (Bereich 1 im oberen Diagramm), d.h. bereits dort sind nicht die tatsächlich vom Kunden gewünschten oder beispielsweise auf Grund von Sicherheitsüberlegungen erforderlichen Eigenschaften des Systems formuliert. Allerdings werden in dieser Phase nur

[1] Die Aussagen dieser Arbeit beziehen sich allerdings nicht auf Software, sondern auf komplette Systeme. Da die Software aber oftmals die wesentlichen Funktionen bereitstellt, können die Aussagen übertragen werden.



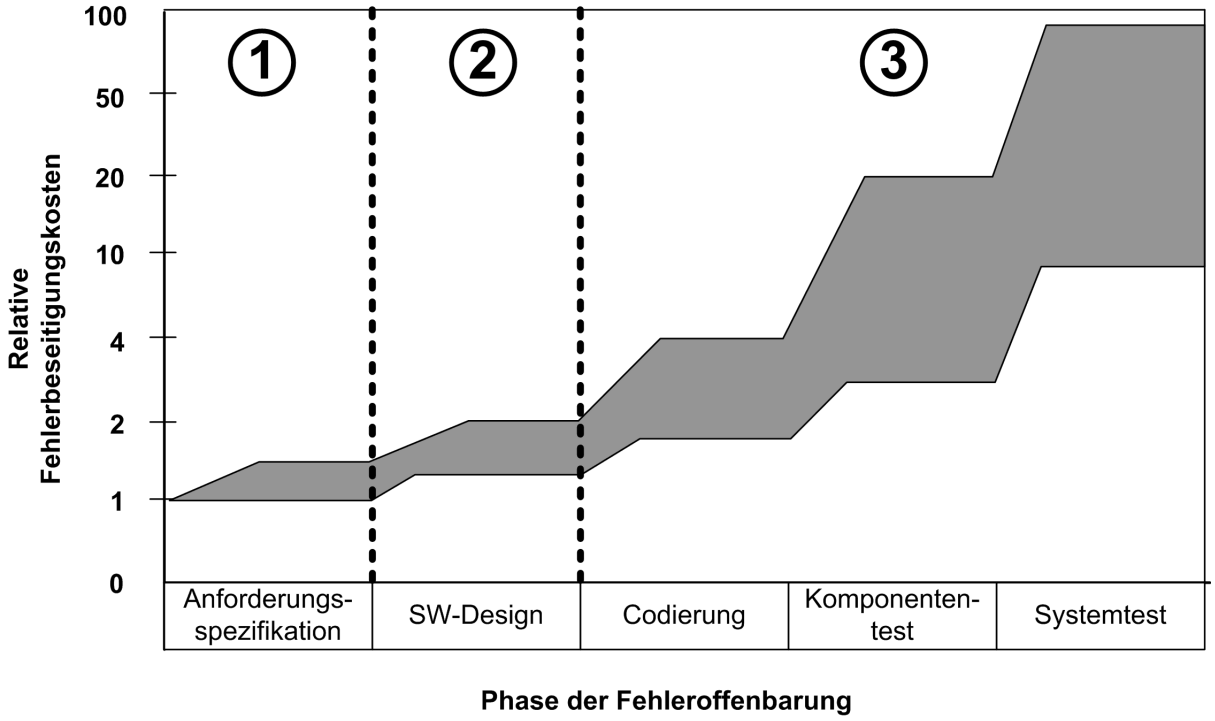


Abbildung 2.1.: Eingebrachte Fehler, gefundene Fehler und Behebungskosten (oben: [STA98], unten: [DOD95])

5 % der späteren Fehler aufgedeckt, so dass diese in nachfolgende Phasen der Systementwicklung (Bereich 2 im Diagramm) verschleppt werden. Dort werden sie zum überwiegenden Teil erst während des Abnahmetests oder sogar erst beim Betrieb des Systems entdeckt (im Bereich 3).

Besonders deutlich werden die wirtschaftlichen Auswirkungen dieses Phänomens, wenn man zusätzlich berücksichtigt, welche Kosten durch die Behebung der Fehler auftreten. Diese steigen deutlich, wenn die Fehlerbehebung in eine späte Lebenszyklusphase fällt. Dieser Zusammenhang wird in vielen Literaturquellen beschrieben, so beispielsweise in einer Publikation des Amerikanischen Verteidigungsministeriums [DOD95, S. 111], und wird im unteren Diagramm in Abbildung 2.1 grafisch dargestellt. Die eingekreisten Zahlen im unteren Diagramm verweisen dabei auf die Entwicklungsphasen im oberen Diagramm und verknüpfen somit die Aussagen beider Grafiken[2].

So lässt sich aus dem Diagramm ablesen, dass ein Fehler, der sich erst beim Systemtest offenbart, zwischen 10 und 100 mal mehr Kosten erzeugt, als ein Fehler, der bereits bei der Erstellung der Anforderungsspezifikation aufgedeckt wird.

Aus diesen Fakten lässt sich daher die folgende Aussage ableiten:

- Während der Erstellung der Anforderungsspezifikation werden - bezogen auf das letztendlich realisierte System - die meisten Fehler eingebracht.
- Diese werden jedoch überwiegend erst sehr viel später im Entwicklungsprozess erkannt und behoben, wobei sie erhebliche Kosten verursachen können.
- Fazit: In der Verbesserung der Anforderungsspezifikation liegt somit ein großes Optimierungspotenzial für den gesamten Lebenszyklus eines Systems.

In den letzten Jahren wurden aus diesem Grund in verschiedenen Arbeiten Konzepte und Ideen vorgestellt, wie die Beschreibung von Systemanforderungen verbessert werden könnte. Diese Arbeiten hatten dabei oftmals die Anpassung bestimmter Beschreibungsmittel, wie z.B. der Unified Modeling Language (UML) zum Inhalt oder haben diese um bestimmte Eigenschaften - wie eine formal definierte Semantik [ARA05] - erweitert. Allerdings erfordern neue Beschreibungsmittel auf Grund ihrer Andersartigkeit auch eine neue Herangehensweise an die Gesamtaufgabe der Spezifikation von Systemanforderungen. Davon betroffen sind sowohl die Struktur und Gliederung der Anforderungsspezifikation an sich, als auch die Prozesse zu deren Erstellung.

Im Folgenden wird daher auf Basis der Erkenntnisse vorhergehender Arbeiten ein Gesamtkonzept entworfen, das erstmalig ein semi-formales, modellbasiertes Beschreibungsmittel mit einer Prozessvorgabe zur Erstellung von Anforderungsspezifikationen kombiniert. Dabei werden die Ergebnisse des durch das Bundesministerium für Bildung und Forschung (BMBF) geförderten Projektes OPRAIL [OPR06] aufgegriffen und weiterentwickelt. Die wesentlichen Eckpunkte dabei sind:

- Nutzung eines modellbasierten Ansatzes für die Analyse- und Erstellungs-Phase der Anforderungsspezifikation, wobei modellimmanente Konsistenz- und Adäquatheitsprüfungen um explizite Verifikations- und Validierungstechniken ergänzt werden
- Berücksichtigung der Erfordernis zur langfristigen Archivierbarkeit von Spezifikationen im Eisenbahnwesen durch die Übernahme der relevanten Modellbestandteilen in einen dokumentenorientierten Spezifikationsrahmen

[2] Auch wenn die einzelnen Phasen in beiden Diagrammen unterschiedlich benannt sind, lassen sie sich zu größeren Entwicklungsschritten zusammenfassen, die durch die eingekreisten Zahlen in den Diagrammen identifiziert werden.

- Beschreibung eines durchgängigen Prozesses, der mit einer informellen Vorphase der Anforderungsentwicklung beginnt und den gesamten Lebenszyklus der Anforderungsspezifikation abdeckt.

Dadurch soll letztendlich die Möglichkeit geschaffen werden, im Eisenbahnumfeld semi-formale, modellbasierte Anforderungsspezifikationen einzuführen und damit die oben genannte Optimierung des gesamten Produktlebenszyklus zu erreichen.

Diese Arbeit gliedert sich in drei Hauptkapitel. In Kapitel 3 wird zunächst die aktuelle Situation beschrieben, darauf aufbauend die Problemstellung analysiert und in Kapitel 4 ein Lösungsweg vorgeschlagen. Kapitel 5 umfasst die Beschreibung einer semi-formalen, modellbasierten Anforderungsspezifikation als eines der beiden Kernelemente des vorgeschlagenen Konzepts. Eine iterativ-inkrementelle Prozessvorgabe zur Erstellung des Anforderungsmodells als zweites Kernelement ist Inhalt des Kapitels 6. In Kapitel 7 wird als Abschluss der erreichte Arbeitsstand dokumentiert und ein Ausblick auf zukünftig mögliche Arbeiten gegeben.

3. Bestandsaufnahme und Identifikation von Mängeln

Die Charakteristik einer Anforderungsspezifikation wird wesentlich durch das verwendete Beschreibungsmittel geprägt. Daher wird in Abschnitt 3.1 zunächst der aktuelle Stand von Wissenschaft und Technik beschrieben, wobei die verfügbaren Beschreibungsmittel für die Spezifikation von Systemen im Mittelpunkt stehen. Weiterhin wird dargelegt, wie diese Beschreibungsmittel zurzeit in verschiedenen Lebenszyklusphasen verwendet werden. Dabei wird auch das Umfeld ihrer Verwendung analysiert und die zugehörigen Methodiken und Prozesse werden berücksichtigt. Auf Basis dieser Analyse werden anschließend die Mängel bei der momentanen Praxis der Anforderungsspezifikation herausgearbeitet. Ausgehend davon wird dann in Abschnitt 4 ein Konzept vorgestellt, wie durch die Verwendung alternativer Beschreibungsmittel und eines Prozessmodells diese Situation verbessert werden könnte.

3.1. Stand von Wissenschaft und Technik

3.1.1. Beschreibungsmittel

In den folgenden Unterabschnitten 3.1.1.1 bis 3.1.1.4 wird dargelegt, welche verschiedenen Beschreibungsmittel überhaupt zur Spezifikation von Systemen[1] zur Verfügung stehen. In dieser Aufstellung ist zunächst keine Aussage darüber enthalten, in welchen Bereichen und in welchen Lebenszyklusphasen die aufgeführten Beschreibungsmittel in der Praxis eingesetzt werden. Dies erfolgt - gegliedert nach den wesentlichen Phasen „Systementwicklung" und „Anforderungserstellung" - in den Abschnitten 3.1.2 und 3.1.3. Die momentane Situation wird anschließend in Abschnitt 3.2 zusammengefasst und bewertet.

3.1.1.1. Natürliche Sprache

Die natürliche Sprache ist das dem Menschen naheliegendste Beschreibungsmittel. Sie einzusetzen setzt kaum Einführungsaufwand und prinzipiell keine besonderen Werkzeuge voraus. Allerdings erfordert auch das Beschreibungsmittel „natürlichen Sprache" Anleitungen zu seiner optimalen Verwendung und ein geeignetes Vorgehen, weswegen sich in der Literatur zahlreiche Beschreibungen von Vorgehensweisen und Ansätzen finden. So enthalten beispielsweise [COC03] und [ALE02] Anleitungen zum Erstellen natürlich-sprachlicher Anforderungsspezifikationen bzw. zur Spezifikation von Anwendungsfällen als Teil einer Anforderungsspezifikation. Darüber hinaus existieren Richtlinien und Normen, die die Struktur von natürlich-sprachlichen Spezifikationsdokumenten regeln, beispielsweise die IEEE-Norm 830 [IEEE830].

[1] „Spezifikation" meint in diesem Zusammenhang allgemein die Beschreibung bestimmter Eigenschaften eines Systems zum Zweck des Informationsaustauschs

Allerdings zeigt die natürliche Sprache bei der funktionalen Spezifikation von sicherheitskritischen Systemen einige immanente Nachteile, die auch durch eine optimale Strukturierung des Erstellungsprozesses und der Dokumente nicht ausgeglichen werden können:

- Natürlich-sprachlich formulierte Dokumente sind nicht eindeutig, weil die Syntax- und Semantikdefinition natürlicher Sprachen immer Interpretationsspielraum lässt.
- Natürlich-sprachliche Dokumentationen komplexer Systeme erreichen oftmals einen großen Umfang, was zu einem hohen Bearbeitungsaufwand führt.
- Natürlich-sprachliche Anforderungsspezifikationen setzen üblicherweise erhebliches Fachwissen voraus, da die durch die Verwendung natürlicher Sprache fehlende Genauigkeit durch viele Annahmen des Lesers implizit kompensiert werden muss.
- Natürlich-sprachliche Dokumentationen lassen sich nicht automatisiert auf Vollständigkeit, Widerspruchsfreiheit und Korrektheit prüfen.

3.1.1.2. Formale Sprachen

Eine Möglichkeit zur exakten und eindeutigen Beschreibung von Systemen besteht in der Verwendung mathematisch/logisch orientierter, formaler Spezifikations- und Beschreibungssprachen. Zu dieser Gruppe gehören - als exemplarische Aufzählung - unter anderen die folgenden Sprachen:

- TLA+ [LAM02], eine von Leslie Lamport entwickelte und auf Temporallogik und Aktionslogik basierende Sprache, die zur Spezifikation von Systemen genutzt werden kann
- Die formalen Sprachen B [SCH01] und Z [IEC13568], die auf den Arbeiten von Jean-Raymond Abrial basieren und die zur Erstellung von Systemspezifikationen verwendet werden können
- Kontrollierte natürliche Sprachen, wie beispielsweise ACE [FUS96], die auf einer syntaktischen und semantischen Einschränkung natürlicher Sprachen (hier: Englisch) basieren. In kontrollierten Sprachen lassen sich Spezifikationen schreiben, die auf den ersten Blick natürlich-sprachlichen Texten ähneln, sich aber auf Grund ihrer speziellen Eigenschaften in formale Spezifikationen überführen lassen und somit ausführbar und durch Test- und Verifikationstechniken überprüfbar werden
- Neben diesen generellen Ansätzen existieren auch Vorschläge für domänenspezifische formale Systembeschreibungssprachen, wie z.B. eine spezifische formale Sprache für die Beschreibung von Leit- und Sicherungssystemen für Eisen- und Straßenbahnen von Haxthausen und Peleska [HAP02]

Die Syntax dieser Gruppe von Beschreibungsmitteln ist üblicherweise stark mathematisch geprägt. Der folgende Ausschnitt aus einer in TLA+ verfassten Systemspezifikation gibt einen Eindruck von der Charakteristik dieser Beschreibungsmittel.

```
invalSet ==
      (*********************************************************)
      (* The set of Inval messages that are generated.         *)
      (*********************************************************)
      IF writing
         THEN { [type  |-> "Inval",
                 cmdr  |-> msg.cmdr,
```

```
            dest  |-> q,
            adr   |-> msg.adr] :
          q \in IF \/ msg.type = "GetExclusive"
                   \/ writer = InMemory
                THEN readers \ {msg.cmdr}
                ELSE (readers \cup {writer}) \ {msg.cmdr} }
    ELSE {}
```

Eine solche Spezifikation ähnelt dabei stark dem Quellcode üblicher Programmiersprachen, was auf Grund des ähnlichen Grades an Formalismus auch zu erwarten ist. Diese recht sperrige Syntax ist gleichzeitig jedoch eine sehr hohe Einstiegshürde für Nutzer ohne formal-mathematische Vorbildung.

3.1.1.3. Grafische Einzelbeschreibungsmittel und kombinierte Ansätze

Ein anderer Ansatz für die Beschreibung von Systemen besteht in der Nutzung formaler, grafischer Beschreibungsmittel, wobei diese üblicherweise nur einzelne Aspekte eines Systems beschreiben, z.B. dessen Verhalten, die Struktur oder Kommunikationsbeziehungen. Zu diesen gehören beispielsweise Petrinetze und Message Sequence Charts. Petrinetze [PET62] eignen sich gut zur Beschreibung von Abläufen, die in diskreten Zeitschritten ablaufen und Nebenläufigkeiten enthalten. Sie lassen sich mathematisch analysieren und können somit einer formalen Prüfung unterworfen werden. Ein anderer Aspekt von Systemen, nämlich die Kommunikationsbeziehungen, lässt sich mit Message Sequence Charts (MSC) darstellen. Dabei handelt es sich um ein hybrides Beschreibungsmittel für das sowohl eine textuelle (MSC/PR), als auch eine grafische Repräsentation (MSC/GR) vorliegt. MSCs sind von der internationalen Telekommunikations-Union (ITU) entwickelt worden und werden von dieser auch weiterentwickelt und gepflegt [ITU04].

Als eine weitere Untergruppe lassen sich grafische Beschreibungsmittel abgrenzen, die mehrere Einzelbeschreibungsmittel integrieren und zu einem durchgängigen Gesamtkonzept vereinen. Dieser Gruppe sind unter anderem die Strukturierte Analyse (SA) von Tom deMarco [MAR79] zuzuordnen, die verschiedene Beschreibungsmittel (Kontextdiagramm, Datenflussdiagramm, Datenwörterbuch, Entscheidungstabellen) methodisch zusammenfasst. Aus dem Zusammenwirken der einzelnen Beschreibungsmittel der strukturierten Analyse kann Code für Programmiersprachen erzeugt werden. Allerdings lässt sich das objektorientierte Paradigma mit der SA nicht umsetzen, weswegen ihre Verwendung in Praxisprojekten rückläufig ist.

Weiterhin existiert mit SCADE ein integriertes Beschreibungsmittel, das auf der formalen Sprache LUSTRE [CPH87] basiert. Es handelt sich dabei um eine Arbeitsumgebung, in der nicht nur Modelle von Systemen entwickelt werden können, sondern auch um eine Test- und Verifikationsplattform und um einen zertifizierten Code-Generator, mit dem sicherheitskritische Software direkt aus dem Modell erstellt werden kann. Der Grad an Integration ist dabei sehr hoch.

3.1.1.4. Modellbasierte Spezifikation mit der Unified Modeling Language (UML) und der Systems Modeling Language (SysML)

Ausgehend von den grafischen Einzelbeschreibungsmitteln wurde unter Einfluss des objektorientierten Paradigmas Mitte der 1990er Jahre eine vereinheitlichte, grafische Beschreibungssprache für Software entwickelt, die Unified Modeling Language (UML). Prinzipiell vereint die UML

Beschreibungsmittel für verschiedene konzeptuelle und dynamische Aspekte eines Systems und basiert dabei auf dem objektorientierten Paradigma, das ein System als eine Menge interagierender Objekte versteht. Die einzelnen Aspekte eines Systems können in der UML durch insgesamt 13 verschiedene Diagrammarten beschrieben werden, die die Struktur eines Systems (z.B. in Klassen- und Komponentendiagrammen) und dessen Verhalten (in Aktivitätsdiagrammen, Zustandsdiagrammen, Sequenzdiagrammen, etc.) erfassen. Umfassende, weiterführende Informationen zu den Diagrammarten, der Syntax und Semantik der UML finden sich in zahlreichen Literaturquellen, z.B. in [RQZ07], weswegen hier nicht weiter darauf eingegangen wird.

Die UML wird durch die Object Management Group (OMG) verwaltet und weiterentwickelt. Aktuell liegt die UML-Spezifikation in der erheblich umfangreicheren Version 2.1.2[2] vor. Die Sprachdefinition untergliedert sich dabei in drei wesentliche Elemente:

- Die *Infrastructure Definition* definiert die internen Bestandteile, aus denen sich die UML zusammensetzt [OMG07-3]
- Die *Superstructure Definition* [OMG07-2] legt die eigentlichen, vom Anwender nutzbaren Sprachelemente auf Basis der Infrastructure Definition fest
- Die vom IBM entwickelte *Object Constraint Language* (OCL), die in die UML-Spezifikation aufgenommen wurde, um bestimmte Zusicherungen, wie z.B. Invarianten in Klassendiagrammen oder Bedingungen in Sequenzdiagrammen ausdrücken zu können [OMG06-1].

Daneben existiert noch eine Spezifikation für den Austausch von grafischen Diagramminformationen mit der *Diagram Interchange Specification*.

Die UML hat mittlerweile weiter Verbreitung im ingenieurwissenschaftlichen und industriellen Umfeld gefunden und wird durch eine fast unüberschaubare Zahl an Werkzeugen, ergänzenden Tools und sonstigen Produkten unterstützt. Allerdings ist sie nach wie vor sehr durch Konzepte aus der Softwareentwicklung geprägt, was für Ingenieure ohne Informatik-Hintergrund abschreckend wirken kann. Diese Ausrichtung zeigt sich auch in den einzelnen Sprachelementen: zwar können diese durch die semantische Umdeutung auch für die Spezifikation von kompletten Systemen verwendet werden, allerdings fehlen dafür wiederum wesentliche Elemente, wie beispielsweise kontinuierliche Stoff- und Informationsflüsse, parametrische Beschreibungen von mathematischen Funktionen und die Einbindung von textlichen Requirements.

Dieser Mangel wurde erkannt und führte zur Entwicklung einer speziellen Sprache für die Spezifikation von Systemen, der auf der UML basierenden *SysML* [OMG07-1]. Sie ist besonders für die Spezifikation der Anforderungen, des Verhaltens, der Struktur sowie der Randbedingungen komplexer Systeme entwickelt worden und zielt weniger auf software-orientierte Problemstellungen ab als die UML. Die SysML eignet sich daher besser, wenn über die konkrete Realisierung eines Systems und die Verteilung der Funktionalität auf Hardware und Software noch nichts bzw. wenig bekannt ist. Da die SysML auf der UML basiert, besteht eine Abhängigkeit zwischen beiden Sprachen. Das Verhältnis zwischen UML und SysML lässt sich dabei durch die folgenden drei Aussagen definieren und ist in Abbildung 3.1 grafisch dargestellt:

- Zahlreiche UML-Konstrukte wurden direkt in die SysML übernommen und werden dort mit gleicher inhaltlicher Bedeutung verwendet (als „UML4SysML"), wurden aber teilweise umbenannt.
- Einige Konstrukte der UML werden für die SysML nicht benötigt

[2] Im Rahmen dieser Arbeit wird jedoch die frühere Version 2.1.1 der UML-Spezifikation verwendet, die zum Zeitpunkt der Modellierungsarbeiten aktuell war. Um Versionskonflikte in Referenzen zu vermeiden, wurde entschieden, auch nach Veröffentlichung der neueren Version 2.1.2 weiterhin die Version 2.1.1 zu verwenden.

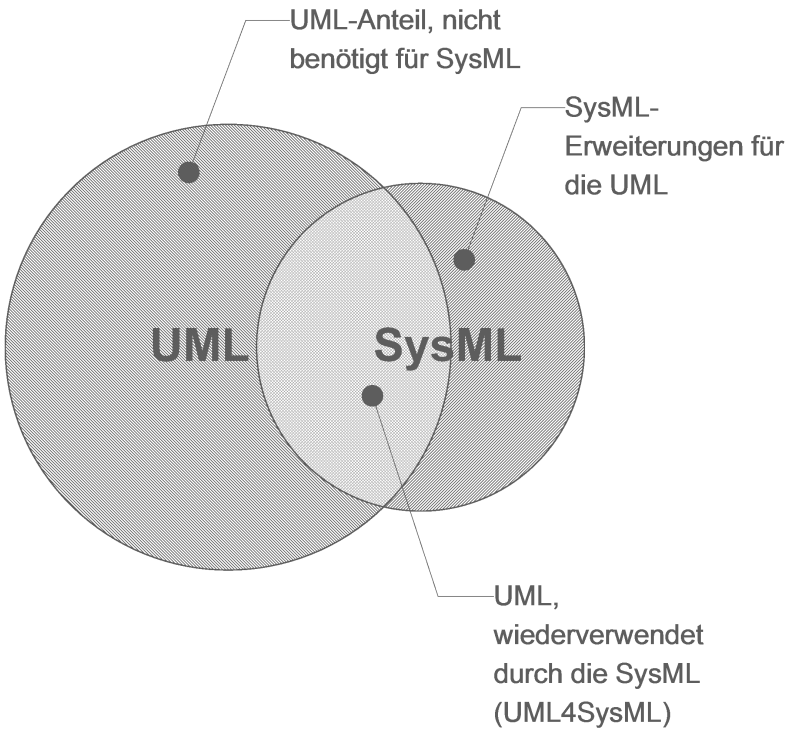


Abbildung 3.1.: Das Verhältnis von UML und SysML

- Einige Konzepte und Elemente wurden in der SysML neu eingeführt - wie beispielsweise der kontinuierliche Fluss von Daten und Materie - oder die Bedeutung von UML-Elementen wurde für den Einsatz in der Systemmodellierung erweitert.

Für Literaturverweise auf die Definition der SysML-Sprachelemente gelten auf Grund dieser Unterscheidungen im Rahmen dieser Arbeit folgende Konventionen: Für Konstrukte der ersten Kategorie (UML4SysML) wird direkt auf die UML-Sprachdefinition verwiesen, da hier auch die SysML-Spezifikation lediglich eine Referenz auf die UML-Spezifikation enthält. Für die beiden anderen Kategorien erfolgt ein Verweis auf die entsprechende SysML-Sprachdefinition in [OMG07-1].

Beide Sprachen gehören zur Gruppe der modellbasierten Beschreibungsmittel. Modellbasiert bedeutet in diesem Kontext, dass durch das Beschreibungsmittel ein vereinfachtes, für einen bestimmten Zweck gedachtes Abbild der Wirklichkeit beschrieben wird. Üblicherweise wird die Vereinfachung gegenüber der Realität dadurch erzielt, dass zum einen nur die für den jeweiligen Zweck relevanten Aspekte im Modell abgebildet werden, zum anderen, dass für diese Aspekte eine hinreichende Vereinfachung vorgenommen wird. Für Anforderungsmodelle besteht der Zweck beispielsweise in der Gewinnung und der Vermittlung von Informationen bezüglich eines zukünftigen Systems. Während der Modellerstellung dominiert dabei die Gewinnung von Informationen, da vormals diffuses und implizites Wissen in eine explizite Form gebracht wird, wobei Wissenslücken auftreten können, die dann entsprechend geschlossen werden müssen. Der Zweck der Informationsvermittlung zeigt sich beispielsweise, wenn anhand eines Modells mit mehreren Parteien über bestimmte Eigenschaften des zukünftigen Systems diskutiert wird.

3.1.2. Momentane Praxis in der Systementwicklung

Nachdem im vorherigen Unterkapitel einige Beschreibungsmittel vorgestellt wurden, wird im Folgenden ein Überblick über die momentane Praxis bei der Software- und Systementwicklung gegeben, wobei insbesondere auf die Verwendung der zuvor aufgeführten Beschreibungsmittel eingegangen wird. Dieser Überblick kann im Rahmen dieser Arbeit nicht wissenschaftlich-charakterisierend sein und erhebt keinen Anspruch auf Vollständigkeit, soll aber aufzeigen, welche Einflussgrößen die Prozesse der Systementwicklung in den letzten Jahren geprägt haben. Der Begriff „Systementwicklung" umfasst in diesem Kontext alle Vorgänge, die auf Seiten eines Herstellers oder Lieferanten ablaufen, um aus einer Anforderungsspezifikation ein fertiges Produkt zu entwickeln.

In dieser Lebenszyklusphase zeichnet sich seit einigen Jahren ein Trend hin zu semi-formalen, objektorientiert-modellbasierten Ansätzen ab. Grund dafür sind zum einen die in vielen Anwendungsbereichen positiven Erfahrungen mit der objektorientierten Analyse (OOA) und dem objektorientierten Design (OOD)[3]. Zum anderen empfehlen zahlreiche Fachnormen, wie beispielsweise das CENELEC-Normenwerk für Bahnsysteme oder diverse militärische Standards [MOD55] die Anwendung formaler bzw. semi-formaler Beschreibungsmittel oder schreiben diese verbindlich vor. Zusätzlich erfordern einige dieser Normen auch die Einhaltung bestimmter Entwicklungsprozesse, weswegen es bei der Systementwicklung schon vor längerer Zeit nötig war, Beschreibungsmittel und Methoden zu integrieren. Im diesem Bereich werden daher zusammen mit den Beschreibungsmitteln häufig auch gut strukturierte Vorgehensweisen und Prozesse eingesetzt (siehe auch Abschnitt 6.1.2), so z.B. der *Rational Unified Process* (RUP) [RUP98] und der *Object Engineering Process* (OEP) [OSK06] oder verschiedene Varianten des *V-Modells* [KBS06]. Neuerdings werden auch vermehrt agile Entwicklungsmethoden verwendet, beispielsweise aus der Familie der *Crystal Methodologies,* die unter anderem in [COC01] beschrieben werden. Diese Integration von Beschreibungsmitteln, Methoden und Werkzeugen wird von Schnieder in [SCN99] mit dem treffenden Begriff „BMW-Prinzip" bezeichnet. Die Kernaussage dieses Prinzips ist dabei, dass für eine effektive Systembeschreibung das Zusammenwirken eines geeigneten Beschreibungsmittels (B), dessen Unterstützung durch Werkzeuge (W) und ein zielgerichtetes Vorgehen in Form einer Methode (M) erforderlich sind. Grafisch lässt sich dieses Prinzip gut durch ineinandergreifende Puzzleteile verdeutlichen (siehe Abbildung 3.2), wobei jedes einzelne Teil eine Komponente des BMW-Prinzips repräsentiert.

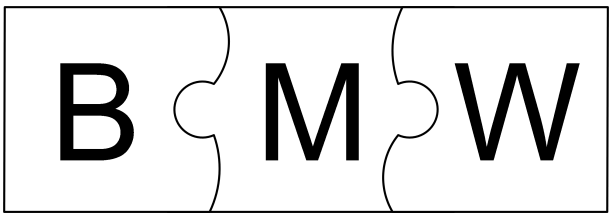


Abbildung 3.2.: BMW-Prinzip

Insgesamt ist bei der Entwicklung sicherheitskritischer, software-lastiger Systeme festzustellen, dass - bezogen auf den Zeitrahmen und den Gesamtumfang eines Projektes - immer mehr Aspekte dem objekt-orientierten, modellbasierte Paradigma unterworfen werden. Knollmann beschreibt in [KNO07] beispielsweise ein durchgängiges Verfahren zur Integration von System-

[3] Wobei objektorientierte Methoden insgesamt nicht frei von Kritik sind und auch nicht in allen Anwendungsfällen eine sinnvolle Wahl darstellen müssen. Sie haben sich aber - ausgehend von der Programmierung grafischer Benutzeroberflächen - in vielen Bereichen des Software-Engineering etabliert.

und Testmodell und erweitert somit die Anwendung semi-formaler Ansätze auch auf den Bereich des Systemtests. Eine Sonderstellung unter diesen integrierten Verfahren nimmt dabei SCADE ein, da hier direkt aus dem Systemmodell Produktivcode erzeugt wird, der direkt auf der Zielplattform ausgeführt werden kann. SCADE ist zusammen mit entsprechenden Prozessvorschriften in der Luftfahrt- und Automobilindustrie verbreitet und erreicht ein bisher ungekanntes Maß an Durchgängigkeit der Entwicklung von ersten Modellen bis hin zum Code.

Im Bereich des Eisenbahnwesens ist die Situation prinzipiell ähnlich wie in anderen Industriezweigen. Allerdings ergab sich gegenüber der früheren Praxis mit der Einführung der CENELEC-Normen ein deutlicher Umbruch, da diese durch den risikoorientierten Ansatz zwar mehr Freiheiten bei der Produktentwicklung erlauben, andererseits aber auch den Einsatz formaler oder semi-formaler Beschreibungsmittel und eines geeigneten Prozesses zur Systementwicklung fordern. Zur Umsetzung der eher vagen Vorgaben aus den entsprechenden Normen wurden - teilweise Hersteller-intern, teilweise in Forschungsvorhaben - verschiedene Prozessmodelle entwickelt.

Ein konkretes Beispiel für einen solchen weitgehend nach dem BMW-Prinzip aufgebauten, aktuellen Entwicklungsprozess für softwarelastige bahntechnische Systeme ist der OPRAIL-Prozess, der zwischen 2004 und 2006 im Rahmen des gleichnamigen Forschungsvorhabens entwickelt wurde. Dieses Vorhaben war Teil der Forschungsoffensive „Software Engineering 2006" des Bundesministeriums für Bildung und Forschung (BMBF) und hatte zum Ziel, CENELEC-konforme Entwicklungsprozesse für bahntechnische Anwendungen zu optimieren. Wesentliche Elemente des OPRAIL-Vorgehens sind dabei nach [OPR06, S. 14]:

> „As defined [...] the main goal of the OPRAIL project is the optimization of CENELEC conformant development processes of safety critical applications for railway systems by means of:
> - Usage of formal and semi formal methodology as basis of a MDD-Process
> - Usage of formal verification during specification of railway systems
> - Usage of tools for automated test case generation and test case execution
> [...]
> - Definition of an UML-subset (Safe-UML)"

Im Vorhaben OPRAIL wurde somit eine Arbeitsumgebung definiert, bei der der Fokus auf der Erstellung eines normenkonformen Entwicklungsprozesses (siehe Abbildung 3.3), der Nutzung semi-formaler Beschreibungsmittel (UML) und der Integration formaler Test- und Verifikationstechniken liegt.

Kernelement von OPRAIL ist eine ausführliche Prozessdefinition, die auf einem iterativ-inkrementellen Vorgehensmodell basiert und den Nutzer bei der Anwendung der verschiedenen UML-Sprachelemente methodisch unterstützt. Die Prozessdefinition und die einzelnen Schritte des Prozessablaufs sind dabei an die einzelnen Lebenszyklusphasen der CENELEC-Normen angepasst, so dass aus dem Prozess heraus konforme Artefakte und Dokumente nach DIN EN 50128 erstellt werden.

Mit dem OPRAIL-Prozess ist es möglich, software-lastige Systeme komplett modellbasiert zu spezifizieren, die entsprechende Software zu entwickeln und zu implementieren und die Korrektheit des Systems durch Test- und Verifikationstechniken sicherzustellen. Allerdings schließt der OPRAIL-Prozess die Erstellung der Systemanforderungsspezifikation explizit aus. Auch hier wird - wie bei vielen System- und Softwareentwicklungsprozessen - angenommen, dass eine vollständige und korrekte Anforderungsspezifikation vorliegt, ohne näher auf deren Erstellung einzugehen. Auch werden anhand der Prozessdefinition keine Anforderungen an die Systemanforderungsspezifikation selbst formuliert, um diese möglichst effizient als Ausgangsdokument für den Systementwicklungsprozess nutzen zu können.

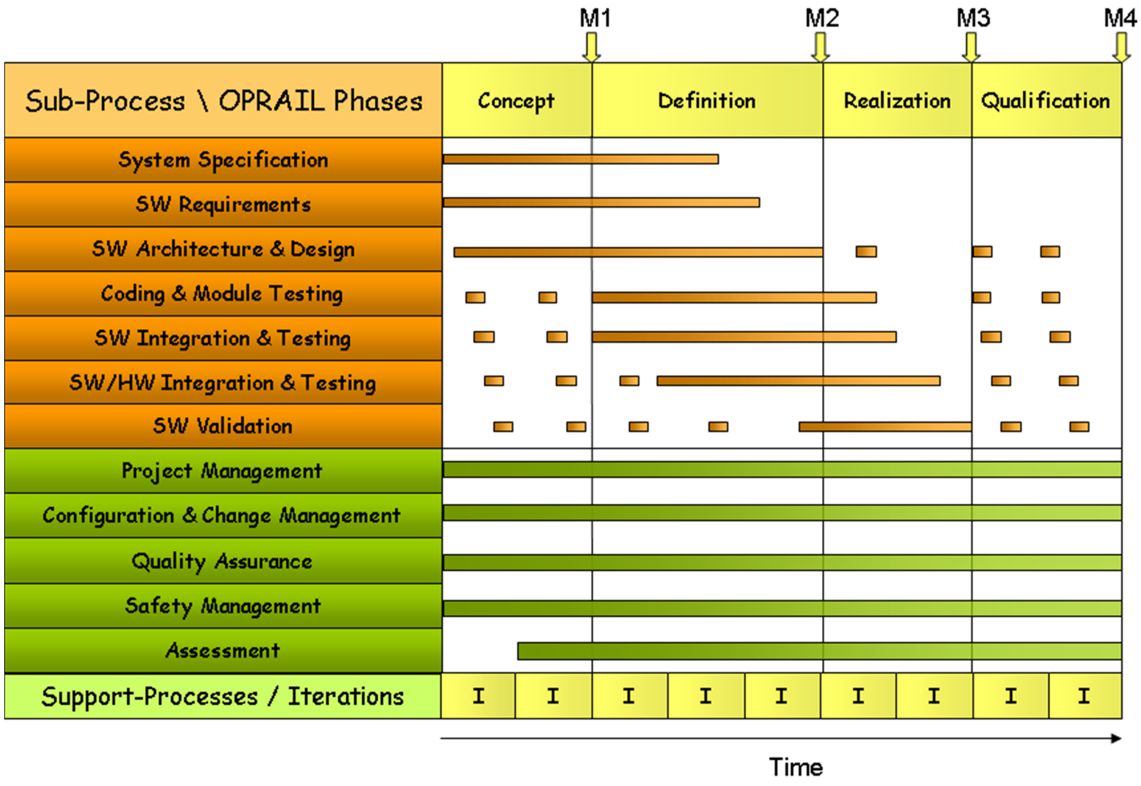


Abbildung 3.3.: OPRAIL-Prozessmodell (nach [OPR06])

3.1.3. Momentane Praxis bei der Anforderungsspezifikation

Nachdem zuvor die augenblickliche Situation bei der System- und Softwareentwicklung vorgestellt wurde, wird in diesem Unterabschnitt die unmittelbar vorgelagerte Spezifikation der Anforderungen beschrieben, wobei der Schwerpunkt auf der Praxis im Eisenbahnwesen liegt. Dort dominiert zum jetzigen Zeitpunkt eindeutig die natürlich-sprachliche Beschreibung von Systemanforderungen in unterschiedlichen Ausprägungen und Qualitätsstufen. Gemeinsam sind allen Dokumenten damit auch die Probleme natürlich-sprachlicher Spezifikationen, wie sie in Abschnitt 3.1.1.1 dargelegt wurden.

Die zugehörigen Spezifikationen lassen sich in zwei wesentliche Klassen einteilen: Ältere Spezifikationen, wie beispielsweise das Lastenheft für elektronische Stellwerke (ESTW) oder für die linienförmige Zugbeeinflussung (LZB) bilden eine dieser beiden Klassen. Charakteristisch dabei ist, dass die Dokumente eher durch historisch gewachsene Strukturen geprägt sind, als durch eine sorgfältige Gliederung der Informationen. Dies führt in vielen Fällen dazu, dass eine Systementwicklung ohne spezifisches Vorwissen und ohne langjährige Einarbeitung in das Lastenheft quasi unmöglich ist. Dadurch wird insbesondere neuen oder ausländischen Auftragnehmern der Zugang zum Markt erschwert. Auch Änderungen und Korrekturen am Lastenheft selbst sind nur mit hohem Aufwand durchführbar, da eine Prüfung auf Widersprüche, Korrektheit und Vollständigkeit nur manuell möglich ist. Dies führt mitunter zu einem Auseinanderdriften zwischen Spezifikation und tatsächlich implementierten Systemen, da Änderungen nur noch auf Seiten der Hersteller dokumentiert werden.

Zur zweiten Klasse zählen jüngere Anforderungsspezifikationen, wie z.B. das Lastenheft für die Implementierung des European Train Control Systems (ETCS) oder funktionaler Lastenhefte

für LST-Modulverträge für die Deutsche Bahn. Bei diesen Dokumenten zeigt sich üblicherweise eine verbesserte Situation. Diese Anforderungsspezifikationen werden oftmals mit einem Requirements Management Werkzeug (z.B. IBM/telelogic DOORS) erstellt und weisen dadurch eine klarere und sauberere Struktur auf, die einfachere Modifikationen und Fehlerkorrekturen erlaubt. Allerdings sind auch hier die eigentlichen Informationen nach wie vor in natürlicher Sprache abgefasst, wodurch die generellen Probleme dieses Beschreibungsmittels bestehen bleiben. Abbildung 3.4 zeigt einen Ausschnitt aus einer solchen Anforderungsspezifikation, in dem bestimmte betriebliche Abläufe als Anforderungen an das System beschrieben werden.

Teilweise enthalten die Anforderungsspezifikationen neben den textlichen Beschreibungen auch Grafiken und Diagramme zur Erläuterung bestimmter Sachverhalte. In der genannten ETCS-Spezifikation werden beispielsweise betriebliche Abläufe ergänzend zum Text zusätzlich durch UML-Aktivitätsdiagramme dargestellt. Die Diagramme haben zwar nur erklärende Funktion, dennoch bleibt diese Funktionstrennung problematisch, weil mit zwei grundverschiedenen Beschreibungsmitteln der gleiche Sachverhalt beschrieben wird. Dies kann insbesondere bei Änderungen zu Inkonsistenzen führen, zumal sich nicht alle Sachverhalte in beiden Beschreibungsmitteln gleichermaßen ausdrücken lassen.

Trotz der geschilderten Dominanz von natürlich-sprachlichen Anforderungsspezifikationen werden vereinzelt auch formale Sprachen (siehe Abschnitt 3.1.1.2) verwendet. So wurde B bereits für Praxisprojekte im Bahnbereich eingesetzt, z.B. bei der Spezifizierung von Steuerungssystemen für fahrerlose U-Bahnen [BDM98]. Auf breiter Front konnten sich formale Sprachen jedoch nicht durchsetzen. Dies liegt insbesondere an der mangelnden Handhabbarkeit dieser Beschreibungsmittel für die jeweiligen Fachexperten, die üblicherweise nicht aus dem mathematisch-computerwissenschaftlichen Bereich stammen. Der Aufwand zum Erlernen der formalen Sprachen stellt für diese Nutzergruppe eine erhebliche Hürde dar, die dem erzielbaren Nutzen einen erheblichen Anfangsaufwand gegenüberstellt.

Aber auch semi-formale Beschreibungsmittel wie die UML konnten sich bis jetzt nur in Teilbereichen der Industrie etablieren. So wird die UML von Flugzeugherstellern für die Spezifikation von Subsystemen eingesetzt, beispielsweise bei EADS. Auch in der Bahnindustrie werden für einzelne Subsysteme und für bestimmte Aspekte von Systemen - wie oben beschrieben - UML-basierte Spezifikationen oder zumindest einzelne Diagrammarten verwendet. Eine ganzheitliche, modellbasierte Spezifikation von Anforderungen findet momentan jedoch nicht statt. Ein Grund dafür ist sicherlich, dass für semi-formale Anforderungsspezifikationen im Bahnbereich zur Zeit kein Standard oder ein Best-Practice-Ansatz existiert, weswegen sich jeder potenzielle Anwender selbst mit den oftmals unbekannten Beschreibungsmitteln auseinander setzen und entsprechende Erstellungsprozesse erarbeiten muss. Im Gegensatz zur Systementwicklung fehlt zurzeit eine praxistaugliche Integration von Beschreibungsmittel, Methode/Prozess und Werkzeugkette nach dem BMW-Prinzip. Zwar existieren einige Ansätze, wie z.B. das Vorgehen „SysMOD", das in [WEI06] beschrieben wird, dieses erfüllt aber nicht die Anforderungen für die Spezifikation von sicherheitskritischen Systemen, die Gegenstand dieser Arbeit sind.

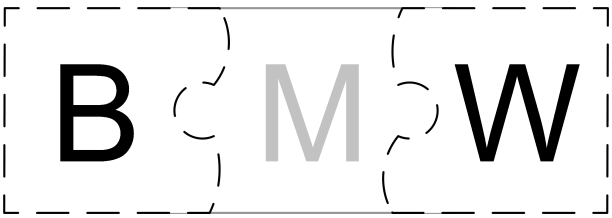


Abbildung 3.5.: Ist-Zustand bei der Umsetzung des BMW-Konzepts

ID	Attribut	Funktionale Anforderungen an die Schnittstelle WAB - WA	Status
FAS_SS_WAB-WA.154	muss	Das Weichenstellsystem stellt die beweglichen Elemente von Fahrwegelementen zur Einstellung des jeweiligen Fahrwegs vom ESTW aus ferngestellt um.	reviewed
FAS_SS_WAB-WA.155	muss	Das Weichenstellsystem kann in ähnlicher Weise für Stellaufgaben an Weichen, einfachen Kreuzungsweichen, doppelten Kreuzungsweichen, Flachkreuzungen und Gleissperren verwendet werden.	reviewed
FAS_SS_WAB-WA.219	muss	Das Weichenstellsystem kann den Zustand "Element in Endlage Minus" einstellen.	reviewed
FAS_SS_WAB-WA.375	muss	Das Weichenstellsystem kann den Zustand "Element in Endlage Plus" einstellen.	reviewed
FAS_SS_WAB-WA.267	muss	Nach Erreichen einer dieser Endlagen muss der Umstellvorgang automatisch beendet werden.	reviewed
FAS_SS_WAB-WA.333	muss	Erreicht das Fahrwegelement die kommandierte Endlage nicht innerhalb einer vorgegebenen Zeitspanne, muss der Umstellvorgang automatisch beendet werden.	reviewed
FAS_SS_WAB-WA.334	muss	Erreicht das Fahrwegelement die kommandierte Endlage nicht innerhalb einer vorgegebenen Zeitspanne, muss eine Störungsmeldung an das ESTW übertragen werden.	reviewed
FAS_SS_WAB-WA.398	muss	Der Weichenantrieb kann in beliebiger Stellung stehen.	reviewed
FAS_SS_WAB-WA.148	-	**2.1.2.2 Zustand der beweglichen Elemente überwachen**	reviewed
FAS_SS_WAB-WA.156	muss	Das Weichenstellsystem muss den Zustand der beweglichen Fahrwegelemente nach Ende des Umstellvorgangs bis zum Beginn eines nachfolgenden Umstellvorgangs feststellen können.	reviewed
FAS_SS_WAB-WA.386	muss	Das Weichenstellsystem muss den Zustand der beweglichen Fahrwegelemente nach Ende des Umstellvorgangs bis zum Beginn eines nachfolgenden Umstellvorgangs kontinuierlich überwachen können.	reviewed
FAS_SS_WAB-WA.176	muss	Ein zusätzlicher Lageprüfer im Weichenantrieb muss in diese Überwachung einbezogen werden können.	reviewed
FAS_SS_WAB-WA.388	muss	Mehrere zusätzliche Lageprüfer außerhalb des Weichenantriebs müssen in diese Überwachung einbezogen werden können.	reviewed
FAS_SS_WAB-WA.389	muss	Bei nicht-auffahrbaren Weichen muss ein Auffahrmelder in diese Überwachung einbezogen werden können.	reviewed

Abbildung 3.4.: Ausschnitt aus textlicher Anforderungsspezifikation [DB09]

Im Kontext des BMW-Prinzips lässt sich die augenblickliche Situation bei der Anforderungserstellung durch die Grafik in Abbildung 3.5 darstellen. Die unterbrochenen Linien bei Beschreibungsmittel und Werkzeug zeigen an, dass für diese Komponenten nur bedingt geeignete Lösungen eingesetzt werden. Das Element „Methode" fehlt vollständig, was durch die hellgrauen Linien angezeigt wird.

3.2. Fazit

Betrachtet man den Lebenszyklus eines Systems von der ersten Idee bis hin zum fertigen System, so zeigt sich, dass sich die Anwendung alternativer, semi-formaler Beschreibungsmittel auf die mittleren und späteren Zeitabschnitte konzentriert. Während im Bereich der Systementwicklung das BMW-Prinzip mit geeigneten Beschreibungsmitteln, Methoden und Werkzeugen gut umgesetzt wird, dominiert bei der Anforderungserstellung nach wie vor die natürlich-sprachliche Prosaform. Setzt man diesen Sachverhalt in Beziehung zur Verteilung von Fehlerquellen in großen Projekten (siehe Abbildung 2.1), so fällt auf, dass die besonders hohe Anzahl eingebrachter Fehler und die Anwendung der natürlichen Sprache in dieser Phase zusammenfallen. Daher liegt der Schluss nahe, dass die Verwendung dieses Beschreibungsmittels zumindest teilweise ursächlich für die hohe Fehlerzahl[4] ist. Zur Optimierung des gesamten Lebenszyklus wäre es somit besonders lohnenswert, auch für die Erstellung der Anforderungsspezifikation auf die bereits in der Systementwicklung bewährten Beschreibungsmittel zurückzugreifen. Dies würde auch den momentan vorhandenen Bruch der Beschreibungsmittel zwischen Anforderungsspezifikation und Systementwicklung beseitigen.

Weiterhin fehlt für die Erstellung von Anforderungsspezifikationen ein geeignetes Prozessmodell. Dieses ist auch insbesondere deshalb nötig, weil die Anforderungsspezifikation üblicherweise eine Aufgabe von Fachexperten des betrachteten Systems ist und nicht von Softwareingenieuren. Allerdings sind Fachexperten meist weniger erfahren in der Anwendung von Beschreibungsmitteln, die ursprünglich aus dem Bereich der Softwareentwicklung stammen. Daher erscheint es zwingend nötig, die Erstellung von semi-formalen Anforderungsspezifikationen durch einen Prozess zu ergänzen und damit den Bearbeiter bei der Erstellung zu führen.

[4] So ist eine natürlich-sprachliche Spezifikation beispielsweise per se nicht automatisiert test- und verifizierbar.

4. Prozessgetriebene, modellbasierte Spezifikation von Systemanforderungen

Nach den Aussagen im vorherigen Kapitel ist die Anwendung alternativer Beschreibungsmittel bei der Anforderungsspezifikation und die Integration in ein Prozessgerüst - mit dem Ziel einer weitgehenden Ablösung der natürlich-sprachlichen Spezifikationen - ein logischer, aber noch ausstehender Schritt.

Auf dieser Basis lässt sich das Ziel der Arbeit formulieren: Die Entwicklung eines auf dem BMW-Prinzip basierenden Konzepts zur Erstellung von Anforderungsspezifikationen unter Nutzung von zeitgemäßen Beschreibungsmitteln, wobei der Schwerpunkt auf der Integration des Beschreibungsmittels und einer an die Aufgabenstellung angepassten Methodik in Form einer Prozessbeschreibung liegt. Dieses Konzept ist dabei auf die Erfüllung der folgenden Ziele ausgerichtet:

- Beschreibung funktionaler Anforderungen in einer ausführbaren Anforderungsspezifikation
- möglichst frühes Testen der funktionalen Anforderungen während der Erstellung
- Integration nicht-funktionaler Anforderungen in das Modell
- Stringente Beschreibung des erforderlichen Vorgehens durch einen Prozess

Im folgenden Kapitel 4.1 wird zunächst dargestellt, auf welche Art von Systemen sich das oben kurz umrissene Konzept anwenden lassen soll. Im nachfolgenden Abschnitt 4.2 werden dann die einzelnen Bestandteile des Konzepts näher beschrieben.

4.1. Abgrenzung des Anwendungsbereichs

Um ein in der Praxis anwendbares Konzept zu entwickeln, muss zunächst abgeleitet werden, auf welche Art von technischen Systemen es anwendbar sein soll. Im Bahnbereich trifft man dabei auf eine sehr heterogene Systemlandschaft, in der sich Systeme und Systemverbünde unterschiedlichster Charakteristik identifizieren lassen. Grundsätzlich sollen für das Konzept im Rahmen dieser Arbeit aktive Systeme betrachtet werden, deren Funktionalität durch elektrische/elektronische Schaltungen oder aber durch Software realisiert wird. Diese übergeordnete Klassifizierung lässt sich nun weiter differenzieren. Dazu werden im Folgenden drei Unterklassen abgegrenzt, die Systeme ähnlicher Charakteristik kapseln:

- Zur ersten Unterklasse gehören dabei funktional eher einfache Systeme mit einem hohen Anteil algorithmischer Verarbeitung und hohen Sicherheits- und Echtzeitanforderungen, die auf Plattformen mit beschränkten Ressourcen ablaufen und zudem mit komplexen Umgebungsbedingungen konfrontiert werden. Zu diesen Systemen gehören beispielsweise Auswerterechner für die Rohsignale der Achszählpunkte, Rechner zur Geschwindigkeits- und Wegerfassung auf Triebfahrzeugen und Systeme zur Signalverarbeitung, z.B. von LZB- oder ETCS-Antennen

- Die zweite Unterklasse umfasst funktional komplexere Systeme, die ereignisgesteuert logik- und zustandsorientierte Aufgabenstellungen bearbeiten, auf eher solide ausgestatteten Hardwareplattformen ablaufen und besser kontrollierbaren Umgebungsbedingungen ausgesetzt sind. Dieser Unterklasse lassen sich beispielsweise abgesetzte Steuerungssysteme für elektrisch ortsbediente Weichen und Bahnübergangsicherungsanlagen, abgrenzbare Komponenten der Stellwerkstechnik (wie etwa Achszählrechner) sowie rechnergesteuerte Komponenten der Fahrzeugtechnik, wie Türsteuerungs- und Zugbeeinflussungssysteme zuordnen.
- Die dritte Klasse umfasst hochvernetzte, sehr komplexe Systemverbünde, die aus zahlreichen Subsystemen bestehen und verschiedenste Aufgaben parallel und verteilt abarbeiten und dabei stark unterschiedliche Sicherheitsniveaus umfassen. Zu dieser Kategorie gehören komplexe elektronische Stellwerke, umfangreiche Systeme zur Betriebsführung - wie beispielsweise Dispositions- und Zuglenksysteme - sowie komplette Fahrzeugsteuersysteme inklusive fahrzeugübergreifender Zugbusse.

Systeme der ersten Unterklasse nehmen durch ihre speziellen Eigenschaften, wie die starke Fokussierung auf algorithmische Fragestellungen und nicht-funktionale Anforderungen wie Ausführungszeiten und Perfomance, eine Sonderstellung ein. Diese erschwert eine gemeinsame Betrachtung mit den Systemen der beiden anderen Unterklassen. Daher wird für den Rahmen dieser Arbeit festgelegt, dass Systeme der ersten Unterklasse nicht im Fokus des im Folgenden entwickelten Konzeptes stehen und somit die Beschreibungsmittel und Prozessabläufe nicht auf deren Besonderheiten abgestimmt werden. Vielmehr soll das Konzept so ausgestaltet werden, dass typische Anforderungen für Systeme der zweiten Unterklasse gut beschrieben werden können. Zusammenfassend lassen sich die Eigenschaften von Systemen, auf die das im Rahmen dieser Arbeit beschriebene Verfahren abgestimmt werden soll, somit folgendermaßen definieren:

- keine rein algorithmische oder signalverarbeitende, sondern reaktive oder service-orientierte Funktionalität
- überwiegend funktionale Anforderungen und weniger nicht-funktionalen Anforderungen
- geeignet für eine objektorientierte Strukturbeschreibung

Das Konzept soll jedoch auch für die Beschreibung von Anforderungen für Systeme aus der dritten Unterklasse geeignet sein. Dazu sollen Möglichkeiten bereitgestellt werden, die hohe Komplexität der Systeme dieser Kategorie hierarchisch in Subsysteme herunterzubrechen, so dass diese den Systemen der zweiten Unterkategorie entsprechen.

4.2. Konzeptbestandteile

Wie in Abschnitt 5.1.1.5 detailliert gezeigt werden wird, ist ein Kernelement für die Erreichung der genannten Ziele die Verwendung semi-formaler, objektorientiert-modellbasierter Beschreibungsmittel. Für die Zwecke der Anforderungsspezifikation im Eisenbahnwesen ist dabei vor allem die Systems Modeling Language (SysML) geeignet.

Damit führt diese Arbeit frühere Arbeiten - wie beispielsweise die Dissertation von Arabestani [ARA05] - fort, in der erstmalig semi-formale Beschreibungsmittel wie die UML als grafischer Überbau für eine formale Semantik verwendet wurden. Dabei lag der Fokus der Betrachtungen jedoch eher auf der formalen Fundierung des Beschreibungsmittels an sich. Der Schwerpunkt dieser Arbeit liegt hingegen auf der Entwicklung eines Gesamtkonzeptes, das eine praxistaugliche Anwendung der Beschreibungsmittel und deren Integration in ein Prozessmodell umfasst.

Dieses Gesamtkonzept wird im Rahmen dieser Arbeit durch zwei wesentliche Aspekte gebildet: Zum einen durch die Definition der Struktur einer *modellbasierten Anforderungsspezifikation* und zum anderen durch einen *Prozess*, der den Anwender durch die Erstellung dieser Anforderungsspezifikation leitet. Die Erstellung des Modells mittels des definierten Prozesses erfolgt dabei unter Zuhilfenahme einer exemplarischen *Werkzeugkette*[1].

Die modellbasierte Anforderungsspezifikation ihrerseits besteht aus drei Komponenten, durch die verschiedene Aspekte des Systems beschrieben werden

- funktionale Modelle zur Beschreibung der funktionalen Systemanforderungen
- Schnittstellen zur Integration nicht-funktionaler Anforderungen
- Nachverfolgbarkeit zu externen, unveränderlichen Dokumenten (z.B. Vorschriften, Gesetze, etc.)

Kernstück ist das *funktionale Modell* des beschriebenen Systems, das mit der SysML modelliert wird. Allerdings genügt die reine Festlegung eines geeigneten Beschreibungsmittels nicht. Modellierungssprachen wie die SysML verfügen über derart viele Sprachelemente, dass eine rein intuitive Nutzung nicht möglich ist: Ohne eine Vorauswahl einer für den jeweiligen Anwendungszweck geeigneten Untermenge der Sprache ist deren Komplexität nicht beherrschbar. Hinzu kommt, dass ohne Vorgaben zur Sprachverwendung unter Umständen unterschiedliche Untermengen der Sprache für die gleiche Aufgabe verwendet werden, was zu untereinander inkompatiblen Modellen führt. Daher sollen im Rahmen dieser Arbeit diejenigen Elemente der SysML in einem Metamodell festgelegt werden, die zur Beschreibung von Systemanforderungsspezifikationen besonders geeignet sind. Ergebnis ist eine Untermenge der gewählten Beschreibungssprache - *SysML(A)* genannt - speziell für die Modellierung von Anforderungsspezifikationen im Eisenbahnwesen.

Die Festlegung der nötigen Untermenge erfolgt dabei nach inhaltlichen Gesichtspunkten in einem Top-Down-Vorgehen: Zunächst werden als erste Gliederungsebene diejenigen Aspekte definiert, die in einem funktionalen Modell enthalten sein müssen, beispielsweise Angaben zu Umgebung, Funktionen oder Verhalten eines Systems. Jeder dieser Aspekte wird dabei durch die Kombination von Inhaltselementen (die beispielsweise Akteure, Kommunikationsflüsse oder übermittelte Informationen repräsentieren) ausgedrückt. Diese werden in einem letzten Schritt dann durch bestimmte Notationselemente des Beschreibungsmittels repräsentiert. Die Gesamtheit der verwendeten Sprachelemente des Beschreibungsmittels bildet dann den Sprachumfang von SysML(A). Diese prinzipiell Struktur zeigt Abbildung 4.1.

[1] Dies geschieht durch die Nennung der im Rahmen dieser Arbeit jeweils verwendeten Werkzeuge innerhalb der Beschreibung einzelner Methodiken

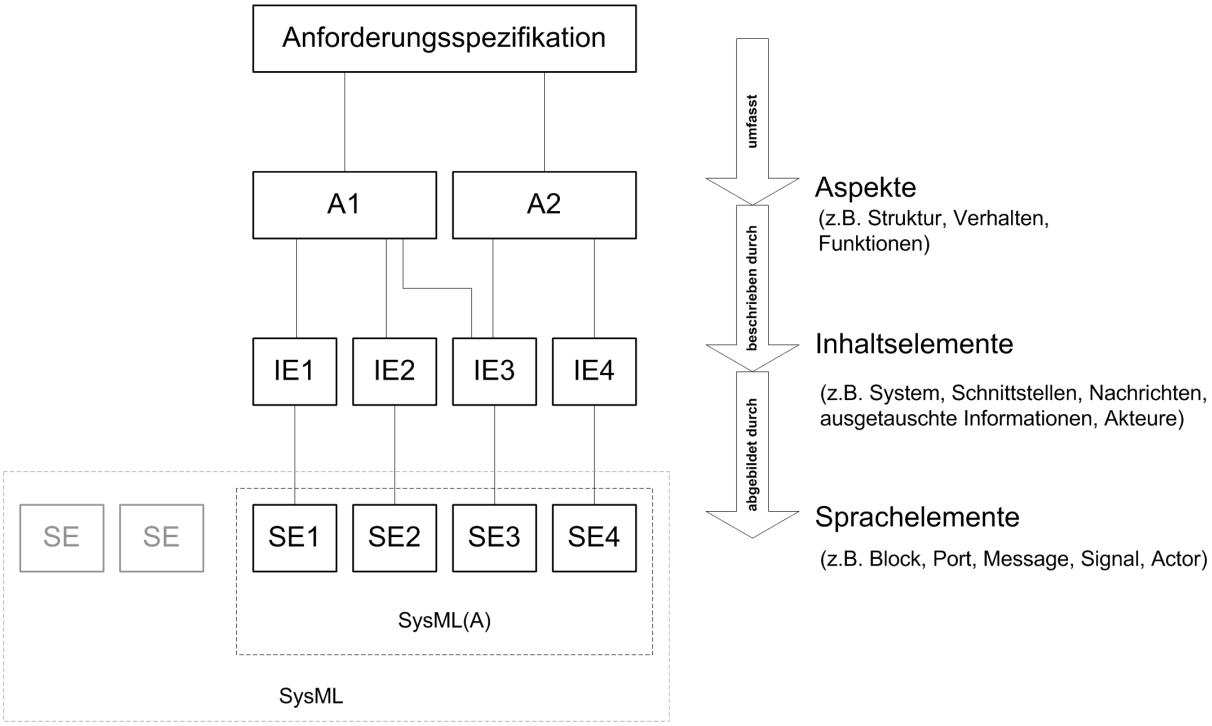


Abbildung 4.1.: Herunterbrechen der Gesamtspezifikation

Um das Modell handhabbar zu machen, müssen darüber hinaus auch Mechanismen zu dessen Strukturierung vorgesehen werden. Dies betrifft insbesondere die Gliederung der Spezifikation in unterschiedliche Detaillierungsgrade und die Organisation der oben beschriebenen Aspekte. Auch diese Informationen umfasst das Metamodell, durch das erstmalig eine Vorlage für alle möglichen Systemanforderungsspezifikationen definiert wird. Dieses Metamodell wird ergänzt durch eine Reihe von Konsistenzbedingungen, die sicherstellen, dass die verschiedenen Aspekte eines Systems zueinander widerspruchsfrei beschrieben werden.

Bei der Beschreibung der beiden übrigen Bestandteile, der *Darstellung nicht-funktionaler Anforderungen* und der *Nachverfolgbarkeit von Anforderungen aus externen, unveränderbaren Dokumenten* kann auf Grund der Heterogenität dieser Informationen oftmals nicht auf die Einbindung als textliche Informationen verzichtet werden. In beiden Fällen müssen somit auch textliche Informationen mit dem funktionalen Modell in Relation gebracht werden. Das Konzept der modellbasierten Anforderungsspezifikation sieht daher Schnittstellen vor, die es einerseits ermöglichen, textliche Informationen direkt in das Modell zu integrieren, sie gleichzeitig mit externen Dokumenten konsistent zu halten und mit geeigneten Werkzeugen zu bearbeiten.

Zweiter wesentlicher Bestandteil dieser Arbeit ist die Spezifikation eines expliziten Prozessmodells für die Erstellung von Anforderungsspezifikationen. Gemäß den Aussagen in Abschnitt 3.1.3 ist eine solche Prozessvorgabe nicht nur eine hilfreiche Zusatzdienstleistung, sondern elementarer, methodischer Bestandteil. Dies gilt umso mehr, als bei potenziellen Erstellern von Anforderungsspezifikationen durchaus Bedenken bestehen, dass sich durch die Anwendung neuer Beschreibungsmittel und Methoden der Aufwand für die Erstellung der Anforderungsspezifikation signifikant erhöht. Dieser Einwand ist berechtigt, denn der Übergang hin zu einer semiformalen Anforderungsspezifikation führt aus zwei wesentlichen Gründen tatsächlich zu einem zunächst höheren Aufwand:

- Das Erlernen und Beherrschen eines neuen Beschreibungsmittel erfordert einen gegenüber

der natürlichen Sprache erhöhten Schulungs- und Einarbeitungsaufwand

- Der höhere Grad an Formalismus und die Test- und Verifikationsphasen zwingen den Bearbeiter zu einer höheren Präzision und vergrößert durch den Zwang zu expliziten Systembeschreibungen den Arbeitsaufwand.

Durch Betrachtung der Folgekosten von Fehlern (siehe Abbildung 2.1) lässt sich dieses Argument jedoch durch die folgenden theoretischen Überlegungen entkräften: Zwar wird der Aufwand zu Beginn des Lebenszyklus eines Systems höher, die zu erwartende Qualitätsverbesserung der Anforderungsspezifikation führt aber potenziell zu weniger Fehlern in späteren Phasen, die dort durch ungleich höhere Aufwände wieder korrigiert werden müssten. Bildlich ausgedrückt besteht die Hoffnung, dass sich der Aufwand, der in frühen Lebenszyklusphasen investiert wird, sich in späteren Lebenszyklusphasen mit Zins und Zinseszins wieder auszahlt[2]. Dennoch wird sich ein neues Vorgehen bei der Anforderungsspezifikation nur dann durchsetzen, wenn es sich praktikabel einsetzten lässt. Und dies ist nur dann der Fall, wenn ein Beschreibungsmittel und ein Prozessmodell geeignet kombiniert werden. Im Rahmen dieser Arbeit wird dies erstmalig für Anforderungsspezifikationen im Eisenbahnwesen beschrieben.

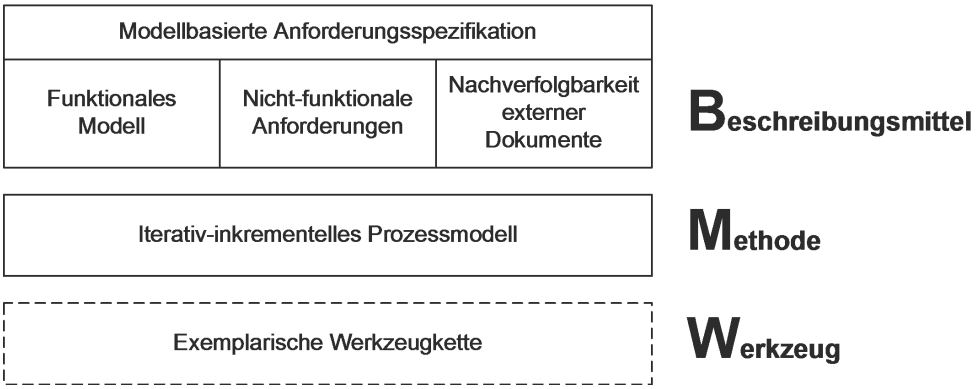


Abbildung 4.2.: Konkrete Umsetzung des BMW-Konzeptes

Das verwendete Prozessmodell wird dabei aus einem geeigneten, allgemeinen Vorgehensmodell abgeleitet. Berücksichtigt werden muss dabei, dass sich die Erstellung einer Anforderungsspezifikation deutlich von der Entwicklung eines Systems unterscheidet. Im Rahmen dieser Arbeit geschieht dies durch die Wahl eines möglichst flexiblen Vorgehensmodells (dynamischer, iterativ-inkrementeller Ansatz), das zudem durch eine Aufgabenorientierung an die bei der Anforderungsspezifikation nur allmählich wachsende Informationsdichte angepasst wird. Der Fokus des Prozesses liegt insgesamt auf der möglichst frühen Ausführbarkeit und Prüfbarkeit der Anforderungsspezifikation. Weiterhin sind in die Prozessbeschreibung Angaben zur Durchführung von Test- und Verifikationsläufen integriert. Letztere sind auf Grund bestimmter genereller Probleme und der daraus resultierenden momentanen Praxisuntauglichkeit im Rahmen dieser Arbeit nur als Platzhalter für zukünftige wissenschaftliche und technische Entwicklungen anzusehen. Dennoch wird bereits durch die Testverfahren eine weitaus intensivere Prüfung des spezifizierten Verhaltens möglich, wodurch sich ein deutlicher Vorteil gegenüber textlichen Anforderungsspezifikationen ergibt. Ergebnis ist ein in vier Phasen gegliederter Prozess, der durch die entsprechende Verwendung von acht Subprozessen und einem iterativ-inkrementellen Prozessablauf eine schrittweise Erstellung des funktionalen Modells erlaubt und dieses nach und nach mit Informationen anreichert.

[2] Diese theoretische Vermutung müsste nach Einführung eines solchen Konzepts empirisch untersucht werden. Eine solche Analyse bietet Raum für interessante weitere Forschungsarbeiten.

Insgesamt ergibt sich somit die in Abbildung 4.2 dargestellte Struktur für das Gesamtkonzept der modellbasierten Anforderungsspezifikation, aufgegliedert nach den Komponenten des BMW-Prinzips. Durch deren Realisierung lässt sich auch für den Bereich der Anforderungsspezifikation das geschlossene Puzzlebild entsprechend Abbildung 3.2 erreichen.

5. Modellbasierte Anforderungsspezifikationen

In diesem Kapitel wird das Anforderungsmodell als eins der Hauptelemente des vorgestellten Gesamtkonzeptes beschrieben, wobei der Fokus auf den strukturellen und inhaltlichen Merkmalen der Anforderungsspezifikation liegt. Der Erstellungsprozess - also das Vorgehen, durch das die hier beschriebene Anforderungsspezifikation erstellt wird - wird im nachfolgenden Kapitel 6 vorgestellt.

Grundsätzlich müssen bei einer Anforderungsspezifikation funktionale und nicht-funktionale Systemanforderungen berücksichtigt werden. Funktionale Anforderungen beschreiben dabei, *was* ein System tun soll, nicht-funktionale Anforderungen hingegen, *wie* und unter *welchen Bedingungen* das System diese Funktionen bereitstellen soll [ROB06, S. 9-10]. Weiterhin existieren für sicherheitskritische Systeme oftmals Anforderungen, die außerhalb des Einflussbereiches des Erstellers der Anforderungsspezifikation liegen, etwa weil sie in Verordnungen, Gesetzen und Richtlinien festgelegt sind. Es muss aber üblicherweise erkennbar sein, welche externen Anforderungen berücksichtig wurden und wie sie den Inhalt der Anforderungsspezifikation beeinflusst haben. Alle drei Komponenten - funktionale Anforderungen, nicht-funktionale Anforderungen und Verweise auf externe Dokumente erfordern auf Grund ihrer unterschiedlichen Charakteristik ein unterschiedliches Vorgehen und werden daher im Rahmen dieser Arbeit in separaten Unterkapiteln behandelt. Das funktionale Anforderungsmodell wird in Abschnitt 5.1 beschrieben, die Integration der nicht-funktionalen Anforderung und die Verweise auf externe Dokumente werden auf Grund ihrer strukturellen Ähnlichkeit zusammen in Abschnitt 5.2 behandelt.

5.1. Funktionales Anforderungsmodell

Das inhaltliche Hauptelement der Anforderungsspezifikation sind die funktionalen Anforderungen, die durch ein geeignetes Beschreibungsmittel ausgedrückt werden müssen. Wie bereits in Abschnitt 4 ausgeführt wurde, eignet sich dafür besonders die SysML. In den folgenden Unterkapiteln wird nun hergeleitet, woraus sich diese besondere Eignung ergibt. Dazu werden in den Abschnitten 5.1.1.1, 5.1.1.2 und 5.1.1.4 zunächst die Anforderungen an ein geeignetes Beschreibungsmittel abgeleitet. Diese Anforderungen werden in Abschnitt 5.1.1.5 gegen die in Kapitel 3.1.1 vorgestellten drei wesentlichen Gruppen von Beschreibungsmitteln evaluiert. Auf Basis dieser Evaluation wird dann beschrieben, warum die letztendlich gewählte SysML das geeignete Beschreibungsmittel darstellt. Deren Anwendung zur Spezifikation der Struktur und des Inhalts des funktionalen Anforderungsmodells wird im Hauptteil dieses Kapitels, den Abschnitten 5.1.3 und 5.1.4, beschrieben. Ergänzt werden diese Ausführungen durch eine Menge an Konsistenzregeln (Kapitel 5.1.5), die die Qualität der im Modell hinterlegten Informationen sicherstellen.

Im Kontext des übergeordneten BMW-Konzeptes betreffen die Aussagen dieses Kapitels den in Grafik 5.1 grau hervorgehobenen Bereich.

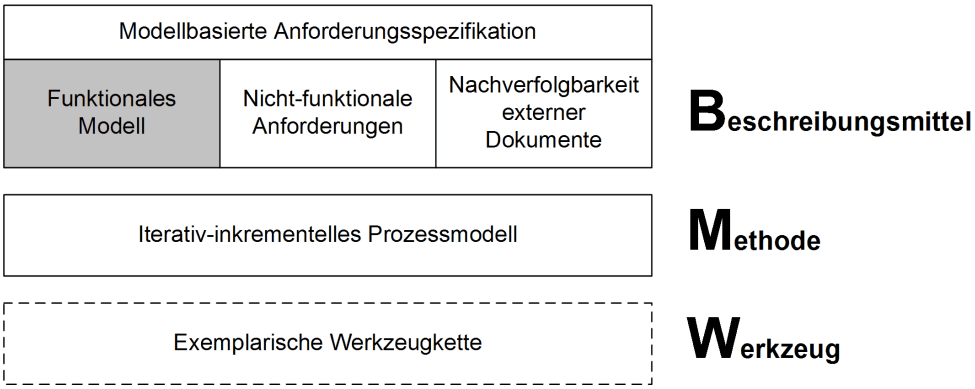


Abbildung 5.1.: Einordnung des funktionalen Modells in den BMW-Kontext

5.1.1. Wahl des Beschreibungsmittels

Um das geeignete Beschreibungsmittel für die Modellierung von funktionalen Systemanforderungen festzulegen, muss zunächst herausgearbeitet werden, welche Anforderungen überhaupt an diesen Teil der Systemanforderungsspezifikation gestellt werden. Im Rahmen dieser Arbeit soll dabei zwischen inhaltlichen, strukturellen und sonstigen Anforderungen unterschieden werden. Die inhaltlichen Anforderungen legen fest, welche Informationen über das zukünftige System durch das Beschreibungsmittel transportiert werden sollen. Durch die strukturellen Anforderungen wird festgelegt, welche Funktionalitäten für Gliederung und Organisation dieser Informationen benötigt werden. Beide Aspekte werden in den folgenden zwei Unterkapiteln erörtert. Das dritte Unterkapitel ergänzt diesen Katalog um sonstige, relevante Anforderungen an das Beschreibungsmittel und fasst anschließend alle Anforderungen in einer Tabelle zusammen. In Abschnitt 5.1.1.3 wird zuvor die Bedeutung des objektorientierten Ansatzes für Beschreibungen in der Fachdomäne des Bahnwesens am Beispiel der Spurplanstellwerke dargestellt.

5.1.1.1. Inhaltliche Aspekte

Für den Umfang von funktionalen Anforderungen existiert dabei ein recht einheitliches Allgemeinverständnis innerhalb der Fachdomäne des Systems Engineering. Dies zeigt sich in verschiedenen Literaturquellen, die Hinweise auf den Umfang der zu beschreibenden Aspekte geben:

- In [WEI06] werden im dort beschriebenen Vorgehen SysMOD verschiedene Arbeitsschritte identifiziert durch die bestimmte Aspekte des Modells erstellt werden. Diese Aspekte sind: der Systemkontext als Darstellung des Systems und der es umgebenden Komponenten [WEI06, S. 52 ff.], die Anwendungsfälle des Systems [WEI06, S. 75 ff.] und die Verhaltensmodellierung der Anwendungsfälle [WEI06, S. 132 ff.].
- Arabestani identifiziert in seiner Dissertation die Systemstruktur [ARA05, S. 22 ff.], das interne Verhalten der Systemkomponenten [ARA05, S. 30 ff.] und deren Interaktion [ARA05, S. 41 ff.] als wesentliche Elemente einer Systembeschreibung
- Hoffmann [HOF08] differenziert in seinem Vorgehen die Modellelemente Anforderungen (in Form von Anwendungsfällen und Sequenzdiagrammen), Systemfunktionsimplementierungen (durch Aktivitätsdiagramme und Zustandsmaschinen) und Systemarchitekturartefakte (als Blockdiagramme).
- In [GAP94, S. 4] werden verschiedene Kriterien zur Bewertung von Methoden zur Anforderungsspezifikation untersucht. Aus einem Teil dieser Kriterien lässt sich auch ableiten,

welche Aufgaben einer Anforderungsspezifikation zugeschrieben werden. Dies sind die Modellierung der Datenstrukturen, der Systemfunktionen, der Datenbewegungen (entspricht den Interaktionen) und von Zeitanforderungen.

Als Schnittmenge dieser Aussagen lässt sich ableiten, dass eine Systemanforderungsspezifikation Aussagen zur Struktur bzw. zum Kontext des Systems, zu den Systemfunktionen, der Interaktion des Systems mit der Umwelt und zum Systemverhalten enthalten muss. Anforderungsspezifikationen im Eisenbahnwesen unterscheiden sich dabei nicht wesentlich von Anforderungsspezifikationen in anderen technischen Bereichen, weswegen sich die allgemeinen Aussagen in oben genannter Literatur auch auf die Fachdomäne der Eisenbahntechnik übertragen lassen. Weiterhin wurde bei der Erstellung der Modelle im Forschungsvorhaben OPRAIL und bei der Bearbeitung des hier verwendeten Beispielmodells eine sehr ähnliche Menge an Aspekten berücksichtigt.

Für eine Systemanforderungsspezifikation ist basierend auf diesen Überlegungen somit die Beschreibung der folgenden funktionalen Aspekte erforderlich:

- Systemabgrenzung - die Abgrenzung des Systems gegen seine Umwelt
- Systemfunktionen - die Darstellung der Systemfunktionen bzw. Systemanwendungsfälle
- Systemverhalten - die Spezifikation des gewünschten Systemverhaltens
- Subsystemarchitektur - je nach Erfordernis eine Spezifikation der Subsystem-Architektur

Weiter muss zur Umsetzung des Gesamtkonzeptes nach 4 im funktionalen Modell eine Schnittstelle zu den nicht-funktionalen Anforderungen und zu externen Dokumenten vorgesehen werden. Dies geschieht gemäß dem in Abschnitt 5.2.3 beschriebenen Prinzip und erfordert eine spezielle Sicht zur Darstellung der extern verwalteten textlichen Anforderungen, die im Rahmen dieser Arbeit *Anforderungsschnittstellensicht* genannt wird.

5.1.1.2. Strukturelle Aspekte

Damit eine funktionale Systemanforderungsspezifikation sinnvoll eingesetzt werden kann, muss nicht nur deren Inhalt, sondern auch deren Struktur bestimmten Anforderungen genügen. Komplexe Systeme erzeugen eine Vielzahl von Spezifikationsartefakten, die ohne eine praktikable Struktur und Organisation zu einer unübersehbaren Datenmenge anwachsen können. Der prinzipielle Vorteil der angestrebten modellbasierten Spezifikation kann durch strukturelle Mängel leicht aufgezehrt werden, wenn die hinterlegten Informationen nicht auf einfache Weise der Spezifikation entnommen werden können. Daher ist es zwingend erforderlich, eine Anforderungsspezifikation sauber zu gliedern und den Nutzern eine praktikable Möglichkeit zur Navigation durch die Spezifikation anzubieten.

Im hier vorgestellten Konzept wird eine zweidimensionale Gliederung der Anforderungsspezifikation vorgeschlagen. Die eine Gliederungsdimension ist der durch *Ebenen* ausgedrückte Detaillierungsgrad, der die Spezifikation in unterschiedliche Betrachtungsebenen unterteilt. Die oberste Betrachtungsebene zeigt das Gesamtsystem, alle untergeordneten Betrachtungsebenen zeigen hierarchisch gegliedert Ausschnitte aus der Subsystemarchitektur. Damit wird die in 4.1 aufgestellte Forderung erfüllt, dass komplexe Systeme zur Anforderungsspezifikation in hierarchisch abhängige Subsysteme gegliedert werden sollen. Die zweite Gliederungsdimension wird durch einzelne *Sichten* gebildet, die jeweils auf einen spezifischen Aspekt des Gesamtsystems fokussieren, beispielsweise auf den Systemkontext oder das Systemverhalten. Beide Dimensionen müssen durch das gewählte Beschreibungsmittel abgebildet werden können.

5.1.1.3. Objektorientierung im Eisenbahnwesen

In Abschnitt 4.1 wurde festgelegt, dass das Konzept zur Anfoderungsspezifikation auf Systeme ausgelegt werden soll, die sich einer objektorientierten Betrachtung unterziehen lassen. In diesem Unterkapitel wird an einem Beispiel gezeigt, dass im Bereich des Eisenbahnwesens der objektorientierte Ansatz zur Beschreibung von reaktiven Systemen gut geeignet und bereits seit längerer Zeit etabliert ist. Betrachtungsgegenstand dieses Beispiels sind Relaisstellwerke, die auf dem so genannten Spurplanprinzip basieren. Dabei handelt es sich um eine prinzipielle Möglichkeit zur Abbildung der Aussenanlage einer Betriebsstelle in der Stellwerkslogik. Die Stellwerkslogik ist dabei

> „[die] zusammenfassende Bezeichnung aller Komponenten eines Stellwerks, die für das Sichern der Fahrwege [. . .] durch Fahrstraßen erforderlich sind." [NAP04, S. 82 ff.]

Nach [PAC04, S. 132] handelt es sich beim Spurplanprinzip um eine geografisch orientierte Fahrstraßenlogik, bei der die physisch vorhandenen Außenelemente einer Betriebsstelle - z.B. Weichen, Gleissperren, Signale und Gleisabschnitte - in der Sicherungslogik als eigenständige Elemente aufgefasst werden. Sie sind entsprechend der realen Gleistopologie miteinander verbunden. Diese Abbildung der physischen Elemente zeigt Abbildung 5.2 für ein einfaches Gleisplanbeispiel.

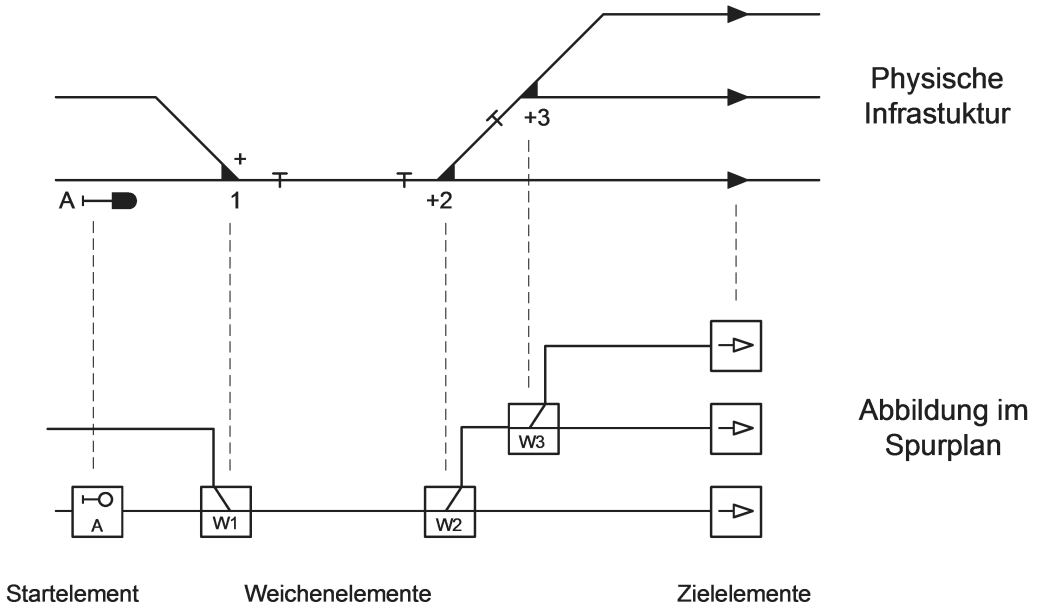


Abbildung 5.2.: Abbildung einer physischen Gleistopologie im Spurplan (nach [PAC04] und [LOR97])

Die technische Umsetzung dieses Prinzip erinnert auffallend an die Paradigmen der objektorientierten Programmierung, obwohl die ersten auf Basis dieses Konzeptes realisierten Stellwerke bereits in den 1960er Jahren in Betrieb gingen, also weit vor der Einführung objektorientierter Methoden in der Informatik. Diese Stellwerke basieren auf einheitlichen Relaisgruppen für jede Art von abzubildenden Außenelementen. Es existieren beispielsweise Relaisgruppen für verschiedene Arten von Weichen, für Signale und Gleiselemente. Alle Relaisgruppen der gleichen Art sind identisch und ähneln somit dem Klassenbegriff der objektorientierten Methoden. Die bei den Relaisgruppen durch entsprechende elektrische Schaltungen definierte innere Funktionalität entspricht dabei dem Verhalten dieser Klassen, wie es bei Software beispielsweise durch

Zustandsautomaten beschrieben werden kann. Die zunächst universellen Relaisgruppen werden dann bei der Verwendung in einem konkreten Stellwerk durch Codierstecker konfiguriert, z.B. zur Definition der zulässigen Abzweiggeschwindigkeit von Weichen. Dies entspricht dem Setzen von initialen Attributswerten im Konstruktur von Software-Klassen. Zur Abbildung des realen Gleisplans werden die einzelnen Relaisgruppen im Stellwerk durch standardisierte Spurkabel entsprechend der äußeren Topologie miteinander verbunden, über die sie auf Basis einfacher Stromflüsse Informationen austauschen. Dies entspricht der Kommunikation unabhängiger Objekte über Nachrichten, wie sie beispielsweise in Sequenzdiagrammen der UML dargestellt wird.

Wie dieses Beispiel zeigt, haben sich die Grundgedanken objektorientierter Konzepte im Eisenbahnwesen sehr früh herausgebildet, weil sie die Charakteristik dieser Fachdomäne gut erfassen. Um von Fachexperten akzeptiert zu werden, liegt es nahe, dass ein Beschreibungsmittel für die Spezifikation eisenbahntechnischer Systeme daher ebenfalls diesen Ansatz unterstützen sollte. Diese Erkenntnis findet sich auch in anderen Veröffentlichung, beispielsweise in [ARG00].

5.1.1.4. Sonstige Anforderungen und Zusammenfassung

Neben den inhaltlichen und strukturellen Anforderungen existieren noch weitere Randbedingungen für das Beschreibungsmittel, die im folgenden hergeleitet werden. Sie beziehen sich nicht direkt auf die Anforderungsspezifikation, sondern vielmehr auf deren Erstellungsprozess und die Integration in den weiteren Systemlebenszyklus. Diese Anforderungen sind:

- Anwendbarkeit des Beschreibungsmittels durch die Zielgruppe - Anforderungsspezifikationen sollten direkt von den Fachexperten für das jeweilige System erstellt werden. Daher muss ein Beschreibungsmittel derart gewählt werden, dass es - eventuell nach einer angemessenen Trainingsphase - von diesen Fachexperten direkt verwendet werden kann. Dies setzt voraus, dass das Beschreibungsmittel in seiner grundlegenden Charakteristik der Denk- und Kommunikationsweise der Fachexperten möglichst nahe kommt.
- Gute Unterstützung durch Werkzeuge - für das Beschreibungsmittel muss eine solide Unterstützung durch Werkzeuge existieren. Diese muss zudem alle Anforderungen für einen Produktiveinsatz hinsichtlich Funktionsumfang, Stabilität und Einbindung in die sonstige Prozess- und Werkzeuglandschaft beim Anwender erfüllen. Idealerweise wird das Beschreibungsmittel zudem durch eine unabhängige Organisation gepflegt und weiterentwickelt.
- Integration des Beschreibungsmittels in den weiteren Prozess - die Erstellung der Systemanforderungsspezifikation steht ganz am Anfang des Systemlebenszyklus. Auf ihr bauen alle weiteren Schritte von der Systementwicklung über die Inbetriebnahme des Systems, dessen Modifikation bis hin zur Außerbetriebnahme auf. Daher sollte ein Beschreibungsmittel gewählt werden, das auch in allen weiteren Lebenszyklusphasen ohne Brüche angewendet werden kann.
- Eignung für die Fachdomäne des Eisenbahnwesens - das Beschreibungsmittel muss für die typischen Charakteristiken von Aufgabenstellungen im Eisenbahnwesen geeignet sein. Wie in Abschnitt 5.1.1.3 dargelegt, bietet sich dabei ein objektorientierter Ansatz an. Diesen muss das Beschreibungsmittel unterstützen.
- Schnittstellen zu formalen Test- und Verifikationstechniken - für eine möglichst fehlerfreie Anforderungsspezifikation bietet es sich an, deren Inhalt durch Test- und/oder Verifikationsverfahren auf Fehler, Inkonsistenzen und Unvollständigkeiten zu prüfen. Das gewählte Beschreibungsmittel soll eine Nutzung dieser Technologien prinzipiell ermöglichen.

- Eignung für das Systems Engineering - Die Anforderungsspezifikation ist ein Dokument auf Systemebene, in der Funktionen und Strukturen ohne Unterscheidung von Hard- und Software beschrieben werden. Das Beschreibungsmittel muss diese Art von Spezifikation unterstützen.
- Nicht-funktionale Anforderungen beziehen sich oftmals nur auf Teile des Systems oder bestimmte Funktionen - es soll daher möglich sein, bestimmte nicht-funktionale Anforderungen explizit einzelnen Entitäten des funktionalen Modells zuzuordnen

In der folgenden Tabelle werden alle Anforderungen zusammengefasst und dienen im nachfolgenden Abschnitt 5.1.1.5 der Ermittlung eines geeigneten Beschreibungsmittels.

Aspekt	Anforderung	Beschreibung
inhaltlich	Darstellung Systemabgrenzung	Beschreibungsmittel muss die Abgrenzung des Systems gegen die Umwelt beschreiben können
inhaltlich	Darstellung Systemfunktionen	Beschreibungsmittel muss die Systemfunktionen beschreiben können
inhaltlich	Darstellung Systemverhalten	Beschreibungsmittel muss das Systemverhalten beschreiben können
inhaltlich	Darstellung Subsystemarchitektur	Beschreibungsmittel muss eine Subsystemarchitektur beschreiben können
strukturell	Organisation von Hierarchien	Beschreibungsmittel muss eine hierarchische Gliederung der Anforderungsspezifikation in Ebenen erlauben
strukturell	Organisation von Sichten	Beschreibungsmittel muss eine fachliche Gliederung der Anforderungsspezifikation in Sichten erlauben
sonstige	Anwendbarkeit durch die Zielgruppe	Domänenexperten müssen das Beschreibungsmittel anwenden können
sonstige	Gute Werkzeugunterstützung	Für das Beschreibungsmittel muss eine gute Werkzeugunterstützung existieren
sonstige	Integration in den weiteren Entwicklungsprozess	Das Beschreibungsmittel muss kompatibel zu den Beschreibungsmitteln im weiteren Systementwicklungsprozess sein
sonstige	Eignung für Fachdomäne Eisenbahnwesen	Für Problemstellungen im Eisenbahnwesen eignet sich der objektorientierte Ansatz. Diesen muss das Beschreibungsmittel unterstützen.
sonstige	Anwendung von Test- und Verifikationstechnologien	Das Beschreibungsmittel soll den Einsatz von Test- und formalen Verifikationstechnologien ermöglichen
sonstige	Beschreibung auf Systemebene	Das Beschreibungsmittel soll Anforderungsspezifikationen auf Systemebene ermöglichen
sonstige	Einbindung nicht-funktionaler Anforderungen	Das Beschreibungsmittel soll Schnittstellen zu nicht-funktionalen Anforderungen vorsehen

5.1.1.5. Diskussion möglicher Beschreibungsmittel für das funktionale Modell

Im Abschnitt 3.1.1 wurden neben der natürlichen Sprache drei wesentliche Gruppen von Beschreibungsmitteln definiert, die im Folgenden gegen die im vorherigen Abschnitt gesammelten Anforderungen evaluiert werden. Diese Gruppen von Beschreibungsmitteln sind:

1. formale Sprachen (siehe Abschnitt 3.1.1.2)

2. grafische Einzelbeschreibungsmittel und kombinierte Methoden (siehe Abschnitt 3.1.1.3)
3. semi-formale Spezifikation mit UML oder SysML (siehe Abschnitt 3.1.1.4)

Berücksichtigt man die gesammelten Anforderungen, so zeigt sich, dass Beschreibungsmittel aus der ersten Gruppe schnell als unbrauchbar für funktionale Systemanforderungsspezifikationen im Bahnbereich ausscheiden. So decken formale Sprachen beispielsweise nicht alle inhaltlichen Aspekte einer Anforderungsspezifikation ab: Während sie sehr gut geeignet sind, das Verhalten eines Systems auszudrücken, ist eine brauchbare Darstellung der Systemfunktionen oder des Systemkontextes kaum machbar. Zudem sind sie für Fachexperten meist nicht intuitiv verwendbar, was ihre Akzeptanz für den Praxiseinsatz deutlich verringert. Sie integrieren sich außerdem schlecht in den weiteren Ablauf der Systementwicklung, da dort - wie bereits festgestellt - vermehrt objektorientierte, modellbasierte Vorgehensweisen und Beschreibungsmittel eingesetzt werden.

Die Beschreibungsmittel der zweiten Gruppe entsprechen auf Grund ihrer grafischen Natur eher den Anforderungen der Domainexperten im Bahnbereich. Sie sind auch ohne tief gehende mathematische Vorkenntnisse zu verstehen und anzuwenden. Allerdings teilen sie mit den Beschreibungsmitteln der ersten Gruppe den wesentlichen Mangel, dass sie entweder nur Teilaspekte eines Systems beschreiben, üblicherweise nicht dem objektorientierten Paradigma entsprechen oder einen Werkzeugbruch in der weiteren Systementwicklung darstellen. Ältere kombinierte Beschreibungsmittel wie beispielsweise die Strukturierte Analyse oder das Strukturierte Design scheitern dabei insbesondere an der fehlenden Objektorientierung, dem Mangel an zeitgemäßen Werkzeugen und der schlechten Integration in heutige Prozesslandschaften. SCADE wird im Produktiveinsatz verwendet und kommt dabei insbesondere in der Luftfahrtindustrie und in der Kerntechnik zum Einsatz, hat aber auch einige Nachteile: So handelt es sich bei SCADE um ein proprietäres Produkt, bei dem man zwingend auf Werkzeuge eines Herstellers angewiesen ist. Die grafische Notation von Scade wird nicht von einer unabhängigen dritten Stelle gepflegt und entwickelt und ist so nur im kommerziellen Produkt Scade Suite nutzbar. Außerdem ist Scade stark auf das Software-Engineering ausgerichtet und damit weniger geeignet für die Beschreibung von Systemen.

Die größte Übereinstimmung mit den Anforderungen ergibt sich für die in der dritten Gruppe vorgestellten Beschreibungsmittel UML bzw. SysML. Die UML wurde in der Vergangenheit auch schon für die Spezifikation von bahntechnischen Systemen verwendet [ARA05, ARG00]. Allerdings ist sie nach wie vor ein Beschreibungsmittel für die Modellierung von Software und weist daher entsprechende Probleme in ihrer Anwendung für die System-Anforderungsspezifikation auf. Diese wurden bei der Entwicklung der SysML gezielt angegangen, wodurch sie besonders gut für die hier beschriebene Aufgabe geeignet ist. Sie ermöglicht die Beschreibung aller inhaltlichen Aspekte durch verschiedene Diagrammarten, stellt Mechanismen für eine Gliederung des Modells in Sichten bereit und erfüllt zudem einen Großteil der sonstigen Anforderungen: So ist sie durch ihre grafische Notation leicht erlernbar, basiert auf dem objektorientierten Prinzip und wird durch eine Vielzahl an unterschiedlichen Werkzeugen unterstützt. Durch Zusatz-Tools sind darüber hinaus auch Test und Verifikation der Modelle möglich. Zudem integriert sie sich gut in gängige Vorgehensweisen zur Systementwicklung, da die Modelle einfach zwischen der SysML und der UML ausgetauscht werden können. Die SysML ermöglicht weiterhin eine semantisch passende Beschreibung für komplette Systeme, auch ohne dass bestimmte Artefakte für diesen Zweck umgedeutet werden müssen. Sie verfügt außerdem über zusätzliche Teilbeschreibungsmittel die an typische Aufgabenstellungen des Systems Engineering angepasst sind. Aus diesen Gründen wird die SysML als Beschreibungssprache zur Beschreibung der funktionalen Systemanforderungen verwendet. Da die SysML zu den modellbasierten Beschreibungsmitteln gehört (siehe 3.1.1.4), ist das Ergebnis der Anwendung ein Anforderungsmodell.

5.1.2. Metamodell und Subset der SysML

Neben der Festlegung des Beschreibungsmittels an sich muss weiterhin definiert werden, welche Notationselemente aus dem gesamten Sprachumfang des Beschreibungsmittels verwendet werden sollen und wie sie zur Erzielung einer bestimmten Aussage zu kombinieren sind. Für diese Aufgabe wird im Rahmen dieser Arbeit das bereits angesprochene *Metamodell* verwendet. Es legt entsprechend dem in Abschnitt 4 beschriebenen Top-Down-Vorgehen fest, aus welchen inhaltlichen Elementen ein Anforderungsmodell bestehen muss, wie diese zueinander in Relation stehen und wie diese Elemente durch SysML-Sprachausdrücke repräsentiert werden. Ein Metamodell kann als Klassendefinition für alle möglichen Anforderungsmodelle verstanden werden. Ein spezifisches Anforderungsmodell entspricht dann einer Instanz dieses Metamodells.

Das in dieser Arbeit verwendete Metamodell ist selbst in SysML spezifiziert und besteht aus SysML-Blockdefinitionen, die verschiedene Bestandteile des Anforderungsmodells repräsentieren:

- *Inhaltselemente* repräsentieren inhaltliche Bestandteile des Anforderungsmodells. Alle Entitäten, die durch das Anforderungsmodell beschrieben werden sollen, müssen durch ein Inhaltselement im Metamodell definiert werden. Soll im Anforderungsmodell beispielsweise der Informationsfluss zwischen Akteuren und System dargestellt werden, muss das Metamodell ein Inhaltselement „Informationsfluss" im Metamodell besitzen. Die Inhaltselemente stehen dabei für ein abstraktes Konzept. Die konkrete Realisierung im Anforderungsmodell durch die SysML wird erst durch die Verknüpfung mit einem *SysML-Sprachelement* festgelegt. Zwischen den Inhaltselementen können Beziehungen existieren. Diese werden durch Assoziationen oder Kompositionen mit entsprechenden Multiplizitätsdefinitionen und Rollennamen realisiert. Inhaltselemente sind im Metamodell mit einer grünen Symbolfarbe gekennzeichnet.
- *Strukturelemente* sind Elemente, die keinen inhaltlichen Beitrag zum Anforderungsmodell leisten, aber zu dessen Strukturierung benötigt werden. Dies sind *Sichten*, *Ebenen*, und *Teilmodelle*, deren Bedeutung und Verwendung in Abschnitt 5.1.4 beschrieben werden. Um anzuzeigen, dass bestimmte Inhaltselemente bestimmten Strukturelementen zugeordnet sind, werden sie durch Assoziationen und Kompositionen miteinander verknüpft. So umfasst das Strukturelement „Systemabgrenzungssicht" beispielsweise die Inhaltselemente „Portdefinition", „SuB" (System unter Betrachtung), „Signal", „Informationsfluss" und „Akteur". Diese Abhängigkeit wird durch Assoziationen zwischen dem Element „Systemabgrenzungssicht" und den genannten Inhaltselementen dargestellt. Die Farbe für Strukturelemente ist dunkel gelb.
- *SysML-Sprachelemente* legen fest, durch welches SysML-Konstrukt ein bestimmtes Inhalts- oder Strukturelement im Anforderungsmodell ausgedrückt werden soll. Üblicherweise verweist dabei ein Inhaltselement auf das zugeordnete Sprachelement. So wird beispielsweise das Inhaltselement „Akteur" in einem Sequenzdiagramm durch das SysML-Sprachelement „LifeLine" repräsentiert. Der Verweis zwischen Modell- und Sprachelement wird durch Assoziationen mit einer „wird dargestellt durch"-Rolle realisiert. Sprachelemente werden mit Symbolfarbe magenta hervorgehoben.
- *SysML-Diagramme* repräsentieren einzelne Diagrammarten der SysML. Durch diese Elemente wird eine Vorgabe bestimmter Diagrammarten für bestimmte Darstellungsaufgaben im Anforderungsmodell möglich. Im Metamodell wird durch dieses Konzept beispielsweise definiert, dass Interaktionen durch Sequenzdiagramme abgebildet werden. Die Verknüpfung zwischen Strukturelementen und SysML-Diagrammen erfolgt über Kompositionen. SysML-Diagramme werden durch eine blaue Farbgebung kenntlich gemacht.

Durch die Trennung von Inhalts- und Sprachelementen im Metamodell wird ein bestimmter Grad von Sprachunabhängigkeit erreicht. Sollte sich in Zukunft eine andere Modellierungssprache als besser geeignet für die Erstellung von Anforderungsmodellen erweisen, kann diese Änderung im Metamodell durch einen Austausch der SysML-Sprachelemente gegen die Sprachelemente des alternativen Beschreibungsmittels umgesetzt werden.

Weiterhin wird durch das Metamodell auch das Subset SysML(A) definiert: Alle SysML-Sprachelemente, die im Metamodell für die Repräsentation der Inhaltselemente benötigt werden, gehören zum entsprechenden Subset. Die Sprachelemente der SysML, die im Metamodell nicht enthalten sind, gehören nicht zu SysML(A).

5.1.3. Modellorganisation des funktionalen Modells

In Abschnitt 5.1.1.2 wurden die Anforderungen an eine geeignete Modellorganisation beschrieben. In den folgenden beiden Unterkapiteln wird dargelegt, wie diese Organisationsstrukturen durch die Gliederung funktionalen Modells und durch die Verwendung entsprechender SysML-Sprachkonstrukte tatsächlich umgesetzt wird.

5.1.3.1. Ebenen

Technische Systeme bestehen oftmals aus einer komplexen Hierarchie von Subsystemen. Um diese Komplexität beherrschbar zu machen, soll das funktionale Modell nach den Aussagen in Abschnitt 5.1.1.2 Mechanismen zur hierarchischen Gliederung vorsehen. Dies geschieht in dieser Arbeit durch die Einführung von *Ebenen.* Eine Ebene umfasst dabei eine einfache zweistufige Hierarchie, in der ein System in mehrere Subsysteme aufgegliedert wird. In der nächst tieferen Ebene werden die Subsysteme zu Systemen, die sich wieder in Subsysteme untergliedern lassen. Mit diesem Prinzip lassen sich beliebig tiefe Hierarchien von Systemen und Subsystemen aufbauen.

In jeder Ebene wird für jedes Teilsystem einmal der gesamte Prozessdurchlauf entsprechend Kapitel 6.7 angewendet, wodurch ein *Teilmodell* mit allen Sichten entsprechend Abschnitt 5.1.3.2 entsteht. Alle Teilmodelle in allen Ebenen ergeben zusammengenommen dann das gesamte funktionale Modell. Bild 5.3 zeigt die sich aus dieser Definition ergebende Gliederung eines Anforderungsmodells mit zwei Ebenen.

Für die Anzahl der Ebenen gibt es keine Vorgabe, da diese je nach Art des Modells und der Aufgabenstellung deutlich schwanken kann. Oftmals wird eine einzige Ebene ausreichen, bei komplexen Systemen mit einer hohen Spezifikationstiefe können jedoch durchaus drei oder mehr Ebenen erforderlich sein.

5.1.3.2. Sichten

Im funktionalen Teil der Anforderungsspezifikation sollen die in Abschnitt 5.1.1.1 aufgeführten inhaltlichen Aspekte eines Systems beschrieben werden. Im Anforderungsmodell wird jeder dieser Aspekte durch eine spezielle *Sicht* auf das jeweilige Teilmodell repräsentiert, wobei die Summe dieser Sichten ein konsistentes Gesamtbild ergeben muss. Eine strukturorientierte Sicht fokussiert beispielsweise auf diejenigen Modellbestandteile, die Aussagen über die Struktur des zu beschreibenden Systems ermöglichen. Eine verhaltensorientierte Sicht hingegen enthält Artefakte zur Verhaltensbeschreibung. Insgesamt werden im Modell die folgenden Sichten abgebildet:

- Systemabgrenzungssicht - stellt die Systemabgrenzung zur Umwelt dar
- Systemfunktionssicht - stellt die einzelnen Systemfunktionen dar
- Szenariensicht - detailliert die Systemfunktionen durch Szenarien
- Systemverhaltenssicht - stellt das Systemverhalten dar
- Subsystemsicht - zeigt erforderlichenfalls die Subsystemarchitektur des Systems
- Anforderungsschnittstellensicht - zeigt die Schnittstellen zu extern verwalteten, textbasierten Dokumenten und zu nicht-funktionalen Anforderungen

Im hier vorgestellten Modellkonzept wird jede Sicht durch die UML/SysML-Konstrukte *view* (Sicht) und *viewpoint* (Standpunkt) abgebildet. Laut SysML-Spezifikation gilt:

> „A view is a representation of a whole system or subsystem from the perspective of a single viewpoint. Views are allowed to import other elements including other packages and other views that conform to the viewpoint." [OMG07-1, S. 27]

Ein *viewpoint* definiert dabei Eigenschaften, die von einer Sicht erfüllt werden müssen und ist definiert als

> „A viewpoint is a specification of the conventions and rules for constructing and using a view for the purpose of addressing a set of stakeholder concerns." [OMG07-1, S. 27]

In Form von einzelnen Textfeldern kann der Zweck eines spezifischen viewpoints definiert werden, wobei für das Anforderungsmodell die Textfelder *purpose* und *concerns* relevant sind. Das Feld *purpose* enthält eine natürlich-sprachliche Kurz-Zusammenfassung der Zielrichtung des betroffenen viewpoints, das Feld *concerns* enthält mehrere Zeichenketten, die die einzelnen Fragestellungen des jeweiligen viewpoints repräsentieren. Eine Sicht muss so aufgebaut sein, dass die jeweiligen Fragestellungen des viewpoints erfüllt werden. Sie wird über eine *conform*-Beziehung mit der view verbunden.

Im funktionalen Modell werden für jede Sicht ein *viewpoint* und eine *view* definiert. Die Sicht kapselt - ähnlich dem *package*-Konstrukt - alle zu dieser Sicht gehörenden Diagramme. Diese Struktur zeigt der Ausschnitt aus dem Struktur-Metamodell in Bild 5.4.

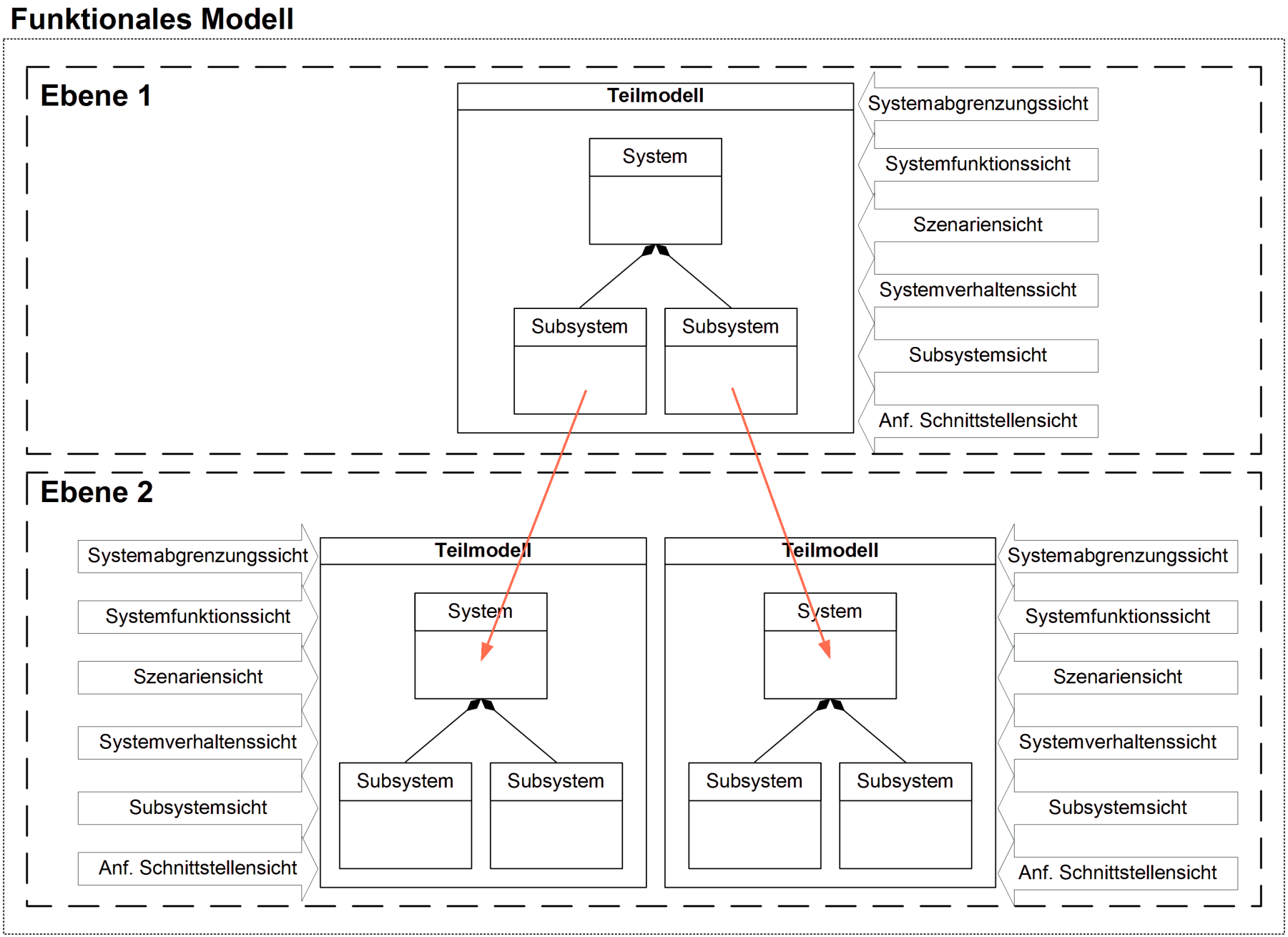


Abbildung 5.3.: Strukturübersicht Anforderungsmodell

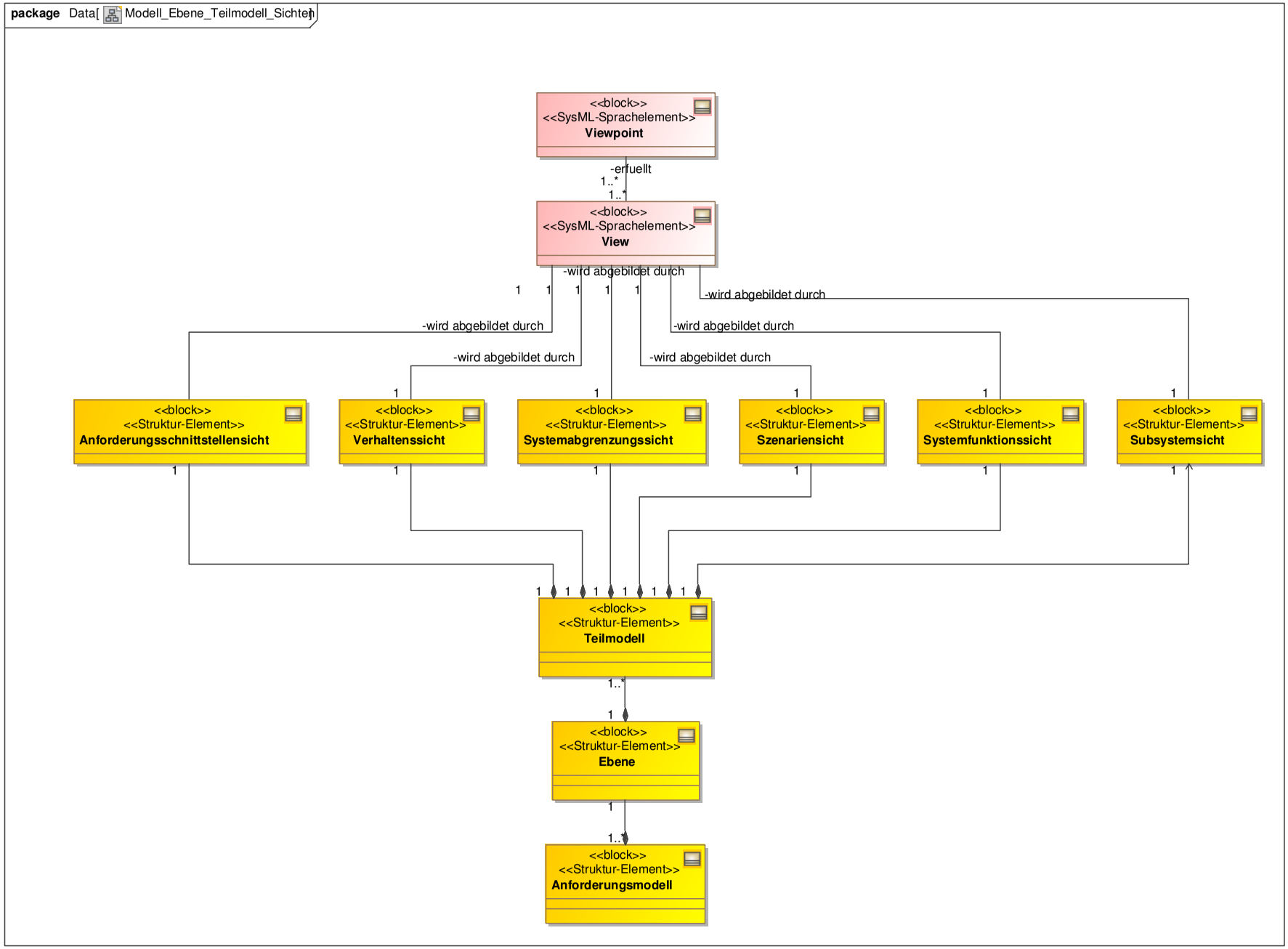


Abbildung 5.4.: Struktur Ebene-Teilmodell-Sicht

5.1.4. Modellinhalt des funktionalen Modells

In diesem Abschnitt werden die in 5.1.3.2 abgeleiteten Sichten inhaltlich analysiert. Ziel der Analyse ist die Festlegung, welche Inhaltselemente zur Beschreibung des jeweiligen Aspekts erforderlich sind. Anschließend wird für jedes Inhaltselement ein geeignetes SysML-Sprachelement identifiziert, wodurch kumulativ der SysML(A)-Sprachumfang aufgebaut wird. Die Arbeitsschritte in den einzelnen Unterkapiteln folgen somit dem bereits beschriebenen Top-Down-Vorgehen. Zusätzlich wird für jede Sicht eine Menge an Konsistenzbedingungen formuliert die bei der Modellerstellung einzuhalten sind. Sie stellen die Konformität der Sichten untereinander und zum Metamodell sicher.

5.1.4.1. Systemabgrenzungssicht

Die Systemabgrenzungssicht zeigt die Grenze zwischen betrachtetem System und dessen Umgebung und legt deren strukturelle Merkmale fest. Die Systemabgrenzungssicht wird dabei vom verwendeten Systembegriff beeinflusst: Erst wenn definiert ist, was unter *dem System* zu verstehen ist, kann eine geeignete Systemabgrenzungssicht modelliert werden. Daher wird im Folgenden zunächst die Auswahl eines passenden Systembegriffs dargelegt, bevor die Umsetzung der Systemabgrenzungssicht im Anforderungsmodell beschrieben wird.

5.1.4.1.1. Begriffsdefinitionen

In den CENELEC-Normen DIN EN 50126, DIN EN 50128 und DIN EN 50129 ist der Systembegriff nicht einheitlich festgelegt. In der DIN EN 50126 heißt es beispielsweise:

> „Ein System kann als Komposition von Subsystemen und Komponenten definiert werden, die zur Gewährleistung der geforderten Funktionalität in geregelter Form miteinander verbunden sind. Die Funktionalität wird den Subsystemen und Komponenten eines Systems zugeschrieben; Verhalten und Zustand des Systems ändern sich mit einer Änderung der Funktionalität eines Subsystems oder einer Komponente. Systeme reagieren im Zusammenwirken mit ihrer Umgebung durch festgelegte Ausgaben (Outputs) auf die erfolgten Eingaben (Inputs).” [EN50126, S. 9]

In der DIN EN 50129 hingegen ist der Systembegriff deutlich knapper gefasst. Demnach ist ein System eine

> „Menge von Teilsystemen, die entsprechend einem Entwurf zusammenwirken” [EN50129, S. 14]

Gemeinsam ist dieser Definitionen eine strukturelle Sicht auf den Systembegriff: Ein System wird als Zusammenstellung von untergeordneten Komponenten verstanden, die zielgerichtet zusammenwirken. Die DIN EN 50126 macht darüber hinausgehend Aussagen über die Zuordnung von Funktionen zu diesen einzelnen Komponenten und über die Interaktion des Systems mit seiner Umgebung.

Ein Systemanforderungsmodell für sicherheitskritische Bahnsysteme muss den CENELEC-Normen genügen. Daher muss sich der verwendete Systembegriff (und daraus resultierend auch die Systemabgrenzungssicht) an oben genannten Definitionen orientieren. Dennoch bietet es sich an, die in den Normen genutzten Systemdefinition zunächst mit verschiedenen, allgemeinen Konzepten der Systemdefinition zu vergleichen. In [ROP99] stellt Ropohl beispielsweise drei generelle Konzepte der Systemtheorie vor:

(a) Das funktionale Konzept betrachtet das System als Blackbox, die auf bestimmte Input-Stimuli mit einer internen Zustandsänderung und einem zugehörigen Output reagiert.

(b) Das strukturelle Konzept beschreibt ein System als die Ganzheit miteinander verknüpfter Elemente. Je nach Art der Verknüpfung dieser Elemente können unterschiedliche Systeme mit verschiedenem Verhalten entstehen.

(c) Das hierarchische Konzept schließlich trägt dem Umstand Rechnung, dass Systeme selbst wieder Subsysteme eines größeren Supersystems sein können. Gleichfalls können Subsysteme in einer tieferen Hierarchie als eigenständige Systeme betrachtet werden.

In [ROP99, S. 77 f.] wird - als Synthese dieser Aussagen - ein System beschrieben als

> „. . . das Modell einer Ganzheit, die (a) Beziehungen zwischen Attributen (Inputs, Outputs, Zuständen etc.) aufweist, die (b) aus miteinander verknüpften Teilen bzw. Subsystemen besteht, und die (c) von ihrer Umgebung bzw. von einem Supersystem abgegrenzt wird."

Vergleicht man diese Systemdefinitionen aus der Literatur mit den Begriffsdefinitionen der CENELEC-Normen, zeigt sich eine durchaus gute Übereinstimmung. Dabei finden sich in der DIN EN 50126 im Wesentlichen die obigen Definitionen (a) und (b), wohingegen die DIN EN 50129 weitgehend nur die Definition von (b) aufgreift.

Um nun vom Systembegriff zur Systemabgrenzungssicht zu gelangen, bietet es sich an, zunächst die Definitionen nach (a) und (c) genauer zu analysieren. Es zeigt sich, dass durch sie implizit bereits eine bestimmte Art von Systemabgrenzung vorgegeben wird. Aus der Blackbox-Definition kann abgeleitet werden, dass es zum einen eine definierte Grenze zur Abgrenzung eben dieser Blackbox geben muss, zum anderen, dass über diese Grenze Inputs in das System hinein- und zugehörige Outputs hinaus gelangen können. Weiterhin kann gefolgert werden, dass das Verhalten des Systems durch die Blackbox gekapselt wird, d.h., dass sämtliche verhaltensbestimmenden Entitäten sich innerhalb der Blackbox befinden. Weiterhin kann aus dem hierarchischen Konzept nach (c) abgeleitet werden, dass ein System auch immer Teil eines größeren, umgebenden Supersystems ist und mit Artefakten dieses Supersystems in Verbindung steht. Diese hierarchische Struktur wird im Anforderungsmodell durch das Konzept der Ebenen (siehe 5.1.3.1) realisiert. Daher muss bei der Systemabgrenzungssicht jedes Teilmodells auch berücksichtigt werden, welche Beziehungen das System zu Elementen des umgebenden Supersystems hat.

Aussage (b) fließt nicht direkt in die Spezifikation der Systemabgrenzungssicht ein. Allerdings gibt das strukturelle Konzept Hinweise darauf, dass zu einem systemorientierten Anforderungsmodell auch eine Beschreibung der Subsystemarchitektur gehört. Dieser Aspekt wir in Abschnitt 5.1.4.5 bei der Beschreibung der Subsystem Whitebox-Sicht aufgegriffen.

Fasst man die obigen Aussagen (a) bis (c) sowie die Definitionen in den CENELEC-Normen zusammen, lassen sich die folgenden Anforderungen an eine geeignete Beschreibung der Systemabgrenzung formulieren:

Die Systemabgrenzung muss

- die Grenze zwischen System und Umgebung
- die hinein fließenden (Inputs) und heraus fließenden Informationen (Outputs)
- die Artefakte des umgebenden Supersystems und deren Beziehungen zum System

darstellen können.

5.1.4.1.2. Modellierung der Systemabgrenzungssicht mit der SysML

Geeignetes Diagramm für die Systemabgrenzungssicht ist das *block definition diagram (Block-Definitions-Diagramm)*. Dabei handelt sich um das SysML-Äquivalent zum Klassendiagramm der UML [OMG07-1, S. 38]. Block-Definitions-Diagramme dienen der allgemeinen Darstellung von strukturellen Systemeigenschaften sowie Hierarchien zwischen Systembausteinen. Abbildung 5.5 auf der nächsten Seite zeigt ein solches Diagramm für einen Beispielfall.

Weiterhin wird an dieser Stelle der Begriff des *Systems unter Betrachtung* (SuB) eingeführt. Es handelt sich dabei um die Bezeichnung desjenigen Artefakts, das das abzugrenzende System im aktuellen Teilmodell repräsentiert. Dieses entspricht damit dem Element, das im weiteren Prozessverlauf genauer spezifiziert werden soll. Zu jedem Zeitpunkt kann dabei nur genau ein Artefakt pro Teilmodell die Rolle des SuB einnehmen. Alle anderen Elemente gehören entweder zum umgebenden Supersystem oder sind Subsysteme des SuB.

Als Modellelement für das SuB wird das SysML-Modellelement *block* (Block) verwendet. Dieses Modellelement entspricht dem der *class* (Klasse) in der UML [OMG07-2, S. 21 ff.], erweitert aber dessen Definition. Entsprechend des breiteren Anwendungsspektrums der SysML muss ein Block nicht zwangsläufig eine Software-Klasse repräsentieren. Ein Block stellt vielmehr ein generelles Konzept dar, um ein System oder Systembestandteil zu beschreiben bzw. dessen Eigenschaften und Funktionen zu kapseln [OMG07-1, S. 31]. Um einen Block als SuB zu spezifizieren, wird er mit einem entsprechenden Stereotyp - z.B. „SuB" - versehen, wodurch er eindeutig als das zu spezifizierende System gekennzeichnet ist.

Für eine bessere optische und inhaltliche Unterscheidung werden die übrigen Elemente, die sich in der Systemumgebung des SuB befinden als *actors* (Akteure) modelliert. Obwohl man das Akteurs-Symbol auf Grund seines Aussehens schnell mit einem menschlichen Bediener assoziiert, soll dieses Modellierungselement hier für alle Arten von externen Systemen verwendet werden. Dies ist auch in Bild 5.5 auf der nächsten Seite zu erkennen: Der zentrale Block „BUe" repräsentiert das zu spezifizierende System. Die Akteure (z.B. „LST-Mitarbeiter", „Schienenfahrzeug" und „Leitstelle") stellen andere Systeme im Kontext des SuB dar.

Zur vollständigen Systemabgrenzung gehört gemäß 5.1.4.1.1 zudem eine Spezifikation der Inputs und Outputs des Systems. Dazu bieten sich die SysML-Artefakte *port* (Port) und *flow port* (Objektflussport) an. Beide Elemente stellen definierte Ein- bzw. Austrittspforten für Informationen an der Systemgrenze zwischen Umgebung und SuB dar. Objektflussports dienen dabei als Interaktionspunkte, über die Daten, Energie oder Stoffflüsse in das System eindringen bzw. das System verlassen können [OMG07-1, S. 59]. Normale Ports hingegen sind geeignet, um Service-orientierte Schnittstellen [OMG07-1, S. 59] zwischen SuB und seiner Umgebung zu beschreiben. Soll beispielsweise ein Taster modelliert werden, über den eine bestimmte Reaktion des Systems ausgelöst werden kann, so ist dafür ein normaler Port das geeignete Modellelement. Ein Anschluss, über den beispielsweise Druckluft in das System gelangen kann, wird hingegen besser über einen Objektflussport spezifiziert. Generell ist nicht vorgegeben, welche Artefakte eines realen Systems durch Ports abgebildet werden. Eine Port kann eine komplexe Softwareschnittstelle repräsentieren, aber auch eine einfache Anzeigelampe oder einen physischen Anschluss für Druckluft. Die Spezifikation, wie ein konkreter Port ausgestaltet wird, erfolgt dann in der Subsystemsicht 5.1.4.5. Für jeden Input in das System und jeden Output aus dem System ist in der Systemabgrenzungssicht demnach ein entsprechender Port festzulegen.

Anschließend muss die Spezifikation der zu übertragenden Informationen erfolgen. Das Vorgehen unterscheidet sich dabei je nach Art des Ports: Objektflussports werden über *flow specifications* (Objektflussspezifikationen) [OMG07-1, S. 64] spezifiziert, normale Ports über *provided* und *required interfaces* (erforderliche und bereitgestellte Schnittstelle) [OMG07-1, S. 69].

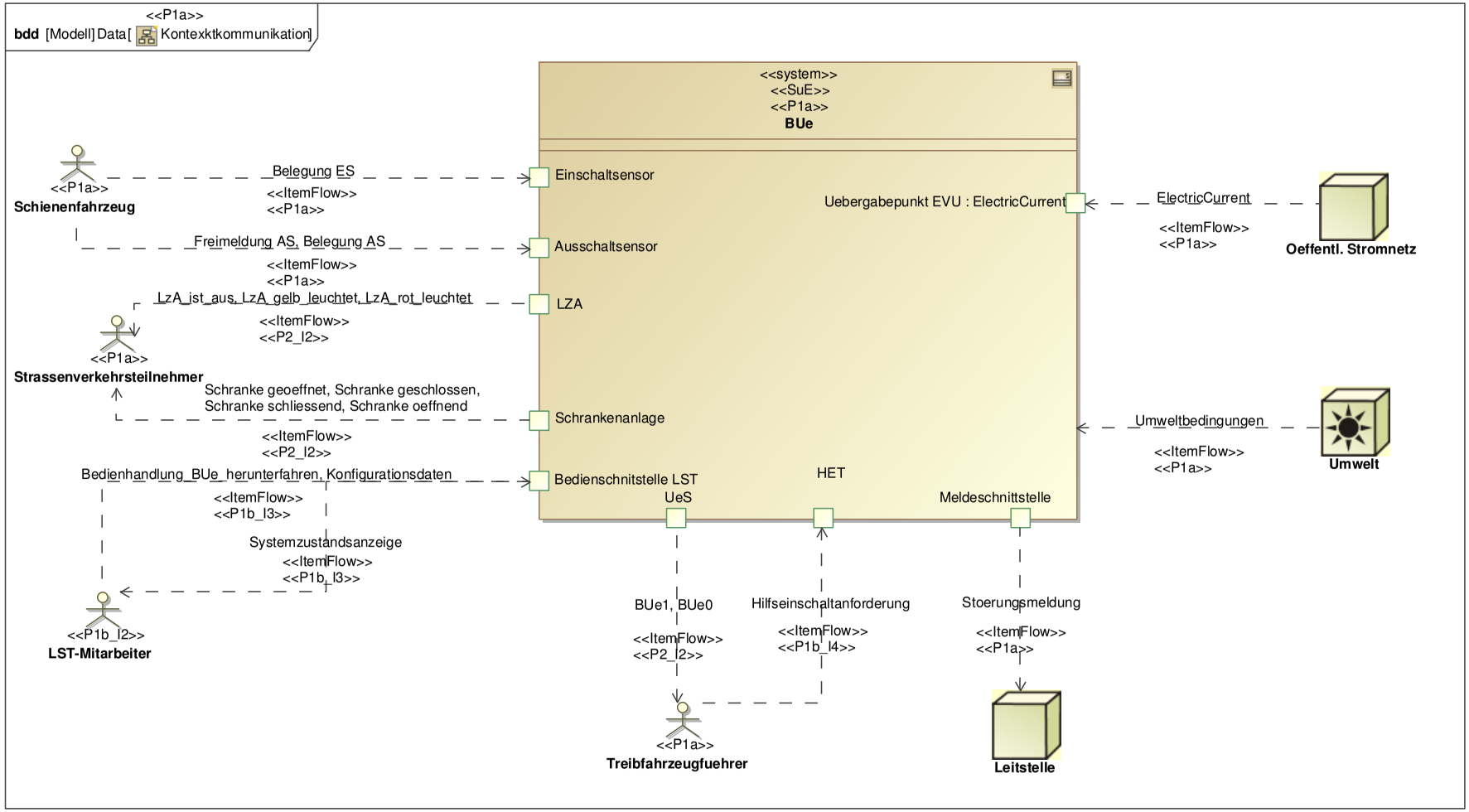


Abbildung 5.5.: Systemabgrenzung durch ein SysML Block-Definitions-Diagramm

Zur vollständigen Beschreibung fehlt nun noch die genaue Spezifikation der Informationsflüsse zwischen dem SuB und den Akteuren. Dies erfolgt durch *item flows* (Objektflüsse) [OMG07-1, S. 64] zwischen Akteur und entsprechendem Port des SuB. Die durch die Objektflüsse übertragenen Informationen lassen sich durch *signals* (Signale) modellieren. Dabei handelt es sich um eine stereotypisierte Klasse, die für den Austausch von Informationen zwischen Objekten verwendet wird [OMG07-2, S. 447].

Ein Ausschnitt aus dem Struktur-Metamodell (Bild 5.6) definiert die in diesem Unterkapitel beschriebenen strukturellen Eigenschaften der Systemabgrenzungssicht mit SysML-Sprachelementen.

Tabelle 5.2 auf Seite 46 zeigt zusammenfassend, welche SysML-Sprachelemente für die Systemabgrenzungssicht verwendet werden.

Inhaltselement	**SysML-Sprachelement**	**Anmerkungen**
Diagrammart	*BlockDefinitionDiagram*	*Entspricht dem Klassendiagramm der UML*
Festlegung des Systems unter Betrachtung (SuB)	Block	Ausgezeichnet durch einen entsprechenden Stereotypen, z.B. „SuB". Nur ein solcher Block kann im Teilmodell enthalten sein.
Andere Systeme in der Umgebung des SuB	Akteur	Auch technische Systeme im umgebenden Supersystem werden durch Akteure dargestellt. Konsistenzkriterium: Die Menge der Akteure muss der Menge der Akteure in der Systemfunktionssicht und der Szenariensicht entsprechen; die Systemverhaltenssicht muss mindestens eine Teilmenge der Akteure enthalten (siehe 5.1.5.2).
Inputs und Outputs über die Systemgrenze	FlowPort bzw. Port	FlowPorts für kontinuierliche oder Stoffflüsse, Ports für Service-orientierte Schnittstellen Konsistenzkriterium: Die Menge der Ports muss der Menge der Ports in der Subsystemsicht entsprechen (siehe 5.1.5.4).
Durch die Ports übertragene Informationen	FlowSpecification bzw. Interfaces	Die Spezifikation der Informationsflüsse kann auch in separaten Diagrammen erfolgen
Informationsflüsse	Item Flows	Konsistenzkriterium: Die Menge der Informationsflüsse zwischen SuB und Akteuren muss der Menge der Informationsflüsse in der Systemfunktionssicht entsprechen (siehe 5.1.5.3).
Ausgetauschte Informationen	Signals	Nutzlast wird durch Attribute modelliert. Konsistenzkriterium: Die Menge der Signale muss der Menge der Signale in der Systemfunktionssicht, der Szenariensicht, der Verhaltenssicht und der Subsystemsicht entsprechen (siehe 5.1.5.1).

Tabelle 5.1.: Modellierungselemente für die Systemabgrenzung

Zu berücksichtigen sind weiterhin die Konsistenzbedingungen zu anderen Sichten im Anforde-

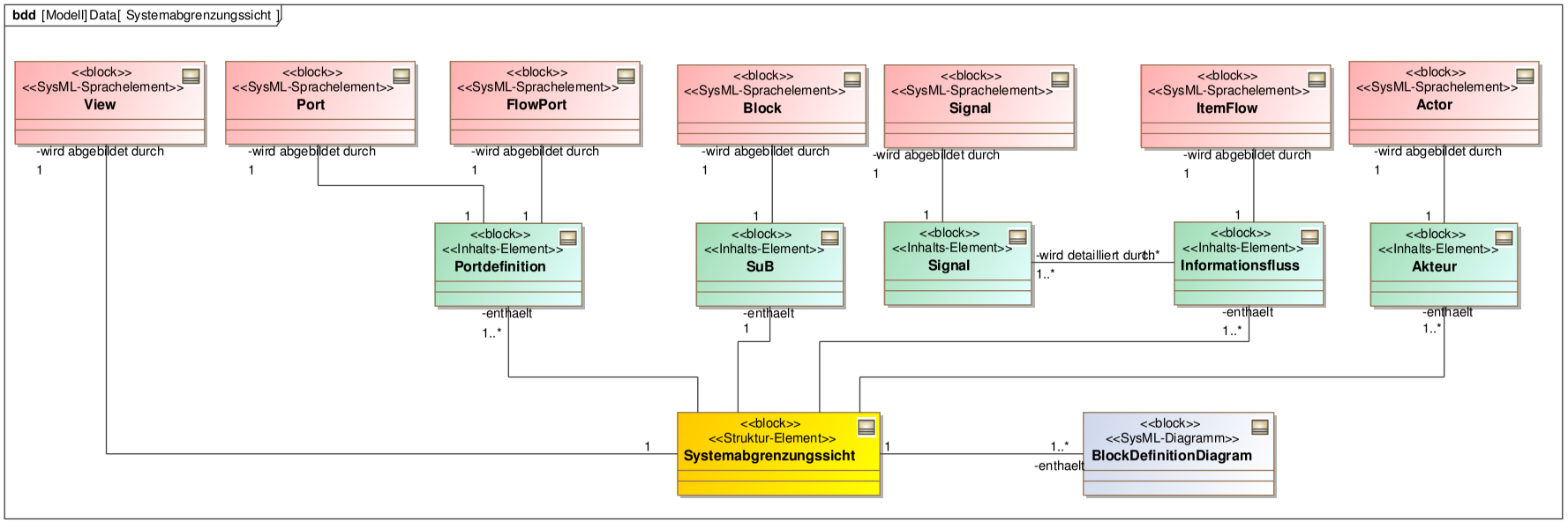


Abbildung 5.6.: Metamodell-Definition für die Systemabgrenzungssicht

rungsmodell und dabei insbesondere die Konsistenzen zwischen Akteuren, Informationsflüssen und Signalen in der Systemabgrenzungssicht und der Systemfunktionssicht (siehe 5.1.4.2). Details dazu können den Unterabschnitten von Kapitel 5.1.5 entnommen werden.

Aus der Systemabgrenzungssicht kann zusammenfassend abgelesen werden:

- wie sich das SuB des aktuellen Teilmodells gegen andere Systeme in seiner Umgebung abgrenzt
- welche Systeme innerhalb des umgebenden Supersystems mit dem SuB in Beziehung stehen
- welche Informationen zwischen dem SuB und der Umgebung über welche Schnittstellen ausgetauscht werden

Damit sind die Forderungen aus 5.1.4.1.1 an die Systemabgrenzung erfüllt.

5.1.4.2. Systemfunktionssicht

In diesem Kapitel wird erörtert, wie die Sicht auf die Funktionen des Systems modelliert werden soll. Zunächst wird dazu hergeleitet, aus welchen Gründen überhaupt eine Sicht auf die Systemfunktionen erforderlich ist, da es für ein ausführbares Modell genügen würde, neben den strukturellen Merkmalen das Verhalten des Systems durch geeignete Beschreibungsmittel zu spezifizieren.

5.1.4.2.1. Gründe für eine funktionsorientierte Sicht

Ein Anforderungsmodell ist zunächst, wie auch eine textliche Anforderungsspezifikation, ein Kommunikationsmittel. Wichtigste Anforderungen an ein solches Kommunikationsmittel sind Eindeutigkeit, Verständlichkeit, Korrektheit und Konsistenz der übermittelten Informationen. Ein Modell, das neben den strukturellen Merkmalen lediglich eine detaillierte Verhaltensbeschreibung durch Zustandsmaschinen oder Aktivitätsdiagramme enthält, ist zwar eindeutig, für einen Leser aber nicht unbedingt verständlich. Erst durch eine explizite Darstellung der Systemfunktionen kann für diesen ein sinnvolles Abstraktionsniveau erreicht werden. Gleichzeitig dienen die Systemfunktionen als Gliederungselement und Navigationshilfe durch das Modell, indem sie als funktionaler Oberbegriff einzelne Aktivitäten oder Zustandsmaschinen zusammenfassen. Diese Abstraktionsebene erleichtert auch die Diskussion des Systementwurfs bezüglich des vorgesehenen Zwecks des Systems, also der Diskussion der Frage, „was das System überhaupt machen soll". Erst wenn dieser Sachverhalt auf abstrakter Ebene hinreichend geklärt ist, kann sich eine detaillierte Verhaltensdefinition durch eine schrittweise Verfeinerung des Systemverhaltens anschließen.

Ein weiterer Nutzen der expliziten Modellierung von Systemfunktionen ist die Wiederverwendbarkeit in der Risikoanalyse. Risikoanalysen sind elementarer Bestandteil des Sicherheitsnachweises für sicherheitskritische technische Systeme und werden von den DIN EN Normen 50126 und 50129 als zwingend notwendiger Arbeitsschritt bei der Systementwicklung festgelegt. Als Ziele der Risikoanalyse nennt Braband in [BRA05, S. 27]:

> „Es ist Aufgabe des Betreibers, eine Risikoanalyse durchzuführen, die darin besteht
>
> - die Funktionsanforderungen für das betreffende System (unabhängig von dessen technischer Ausführung) festzulegen
> - systemrelevante Gefährdungen zu identifizieren

- die Folgen von Gefährdungen zu analysieren
- sicherzustellen, dass das sich ergebende Risiko tolerabel ist und
- die tolerierbaren Gefährdungsraten (THR) abzuleiten."

Besonders relevant für diese Arbeit sind dabei die ersten beiden Punkte „Festlegung der Funktionsanforderungen" und „Identifikation der Gefährdungen": Für die Festlegung der Funktionsanforderungen (erster Punkt) ist eine Systemfunktionssicht direkt hilfreich, aus der die benötigten Informationen unmittelbar entnommen werden können. Ebenso ist eine saubere Funktionsdefinition des Systems wichtige Grundlage für die Identifikation der relevanten Gefährdungen (zweiter Punkt). Dies geschieht üblicherweise in einem zweistufigen Prozess, der aus einer empirischen und einer kreativen Arbeitsphase besteht [BRA05, S. 32]. Während in der empirischen Phase beispielsweise Unfallanalysedaten ausgewertet werden, werden in der kreativen Phase mögliche Gefährdungen durch strukturierte Analysen herausgearbeitet. Ein gängiges Werkzeug für eine solche Analyse sind Fehlerzustandsart- und Fehlerauswirkungsanalysen (*failure mode and effects analyses*, *FMEA*). Der zugehörigen Norm IEC60812 lässt sich entnehmen, dass ein wichtiger Arbeitsschritt in einer FMEA die Ableitung einer funktionalen Struktur des Systems ist [IEC60812, S. 20]. Diese Aufgabe wird deutlich erleichtert, wenn bereits im Anforderungsmodell eine nachvollziehbare, logische Strukturierung der Systemfunktionen vorgenommen wurde.

Es zeigt sich somit, dass eine dedizierte Aufnahme einer Systemfunktionssicht in das Anforderungsmodell nicht nur diesem selbst zugute kommt, sondern auch verschiedene, umgebende Prozesse unterstützen kann.

5.1.4.2.2. Begriffsdefinitionen

Bevor nun die Einzelheiten der Systemfunktionssicht vorgestellt werden, sollen zunächst die verwendeten Begriffe - Funktion, Szenario und Anwendungsfall - für die weitere Verwendung in dieser Arbeit definiert werden. Dies ist erforderlich, da für eben diese Begriffe in verschiedenen Kontexten voneinander abweichende Bedeutungen existieren. Die für diese Arbeit relevanten Begriffsdefinitionen werden ausgehend von der SysML/UML-Spezifikation und den CENELEC-Normen entwickelt. Grundlage ist der Funktionsbegriff in der DIN EN 50129. Dort wird der Begriff Funktion beschrieben als „Art von Aktion oder Tätigkeit, durch die ein Produkt seinen beabsichtigten Zweck erfüllt [EN50129, S. 11]". Diese Definition ist weitgehend deckungsgleich mit der der *use cases* (Anwendungsfalls) im Sinne der UML/SysML. In der entsprechenden Spezifikation wird ein Anwendungsfall beschrieben als

> „A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system. [. . .] Each use case specifies a unit of useful functionality that the subject provides to its users." [OMG07-2, S. 590]

Insbesondere der letzte zitierte Satz entspricht inhaltlich weitgehend der Definition aus der DIN EN 50129. Auf Grund der Ähnlichkeit dieser Definitionen werden die Begriffe Funktion und Anwendungsfall (use case) im Folgenden synonym verwendet.

In der UML-Spezifikation heißt es weiterhin

> „A use case can include possible variations of its basic behavior, including exceptional behavior and error handling." [OMG07-2, S. 590]

Daraus lässt sich die Existenz einer weiteren Gliederungsebene unterhalb der Anwendungsfälle herleiten: Jeder Anwendungsfall enthält verschiedene Variationen des durch ihn repräsentierten Verhaltens. Dieses Konzept wird für den Rahmen dieser Arbeit insofern übernommen, als dass

zu jedem Anwendungsfall eine Menge an Szenarien definiert wird, die den Anwendungsfall sowohl unter Regelbedingungen als auch für Fehlerfälle spezifizieren. Der Begriff *Szenario* wird somit für eine spezielle Ausprägung eines Anwendungsfalls unter spezifischen Randbedingungen verwendet.

5.1.4.2.3. Modellierung der Systemfunktionssicht mit der SysML

Die Modellierung der Systemfunktionssicht erfolgt durch das UML/SysML-Konstrukt der *use cases* (Anwendungsfälle). Zugehöriges Diagramm ist das *use case diagram* (Anwendungsfalldiagramm). Diese Diagrammart stellt die Anwendungsfälle für ein System und die Beziehungen zwischen den Anwendungsfällen und den Akteuren im Systemumfeld dar. Die wesentlichen Modellelemente sind die einzelnen Anwendungsfälle, die Akteure und die Darstellung der Systemgrenze. Für jedes Teilmodell innerhalb des Anforderungsmodells existiert eine eigene Systemfunktionssicht, die sich auf die Anwendungsfälle des SuB in diesem Teilmodell bezieht. Ein solches Diagramm zeigt beispielhaft Abbildung 5.7 auf der nächsten Seite.

Dabei ist zu beachten, dass das Ergebnis des entsprechenden Prozessschrittes (siehe Kapitel 6.3.2) ein hierarchischer Baum von Anwendungsfällen sein kann. Dies liegt im Verfahren begründet, das sich für die Ableitung der Anwendungsfälle bei komplexen Systemen anbietet. Für die Systemfunktionssicht sind allerdings nur diejenigen Anwendungsfälle relevant, die sich auf der untersten Hierarchieebene befinden, also die „Blätter" des Hierarchiebaumes bilden. Daher wird zur besseren Übersicht empfohlen, diese in einem separaten Use-Case-Diagramm darzustellen (Bild 5.7).

Im Anwendungsfalldiagramm repräsentiert jedes Oval einen *use case* (Anwendungsfall), der entsprechend der Begriffsdefinition in 5.1.4.2.2 eine Funktion des Systems repräsentiert. Über Assoziationen ist der Anwendungsfall mit denjenigen Akteuren verbunden, die die entsprechende Funktion vom System einfordern oder eine Ausgabe des Systems weiterverarbeiten. Dabei muss aus Konsistenzgründen sichergestellt sein, dass alle Akteure, die als Systemumgebung festgelegt wurden (siehe 5.1.4.1), auch in den Diagrammen der Systemfunktionssicht enthalten und mit mindestens einem der Anwendungsfälle des Systems assoziiert sind. Lässt sich ein bestimmter Akteur mit keinem der Anwendungsfälle assoziieren, bedeutet dies, dass er offenbar keinen Einfluss auf das Systemverhalten hat und somit auch im Anforderungsmodell nicht erforderlich ist. Ebenso repräsentiert ein Anwendungsfall ganz offensichtlich keine für die Umgebung wichtige Systemfunktion, wenn er keine Verbindung zu den Akteuren besitzt. Damit ist der Anwendungsfall entbehrlich. Alle für die Systemfunktionssicht relevanten Konsistenzbedingungen sind in Kapitel 5.1.5 formal definiert.

Als Ergänzung zu den gängigen Modellierungspraktiken werden in der Systemfunktionssicht auch Angaben zu den Informationsflüssen im Anwendungsfalldiagramm dargestellt. Diese laufen entlang der Assoziationen entweder in den Anwendungsfall hinein oder aus dem Anwendungsfall heraus, was durch entsprechende schwarze Pfeile an den Assoziationen dargestellt wird. Hineinströmenden Informationen bedeuteten, dass der Anwendungsfall diese Informationen entweder für seine Funktionalität benötigt oder auf die eingehende Information auf eine bestimmte Weise reagieren soll. Letzteres trifft vor allem zu, wenn es sich bei den eingehenden Informationen um Serviceanforderungen (typischerweise über einen Service-Port) handelt. Ausgehende Datenflüsse stellen Informationen dar, die im Anwendungsfall generiert wurden und den jeweiligen Akteuren zur Verfügung gestellt werden. Sprachelemente für die Modellierung der Datenflüsse sind *ObjectFlows* (Objektflüsse) entlang der Assoziationen und *signals* (Signale).

Für die Informationsflüsse gelten dabei ganz ähnliche Konsistenzregeln wie für die Akteure: Zu jedem Informationsfluss im Anwendungsfalldiagramm muss ein entsprechendes Gegenstück in

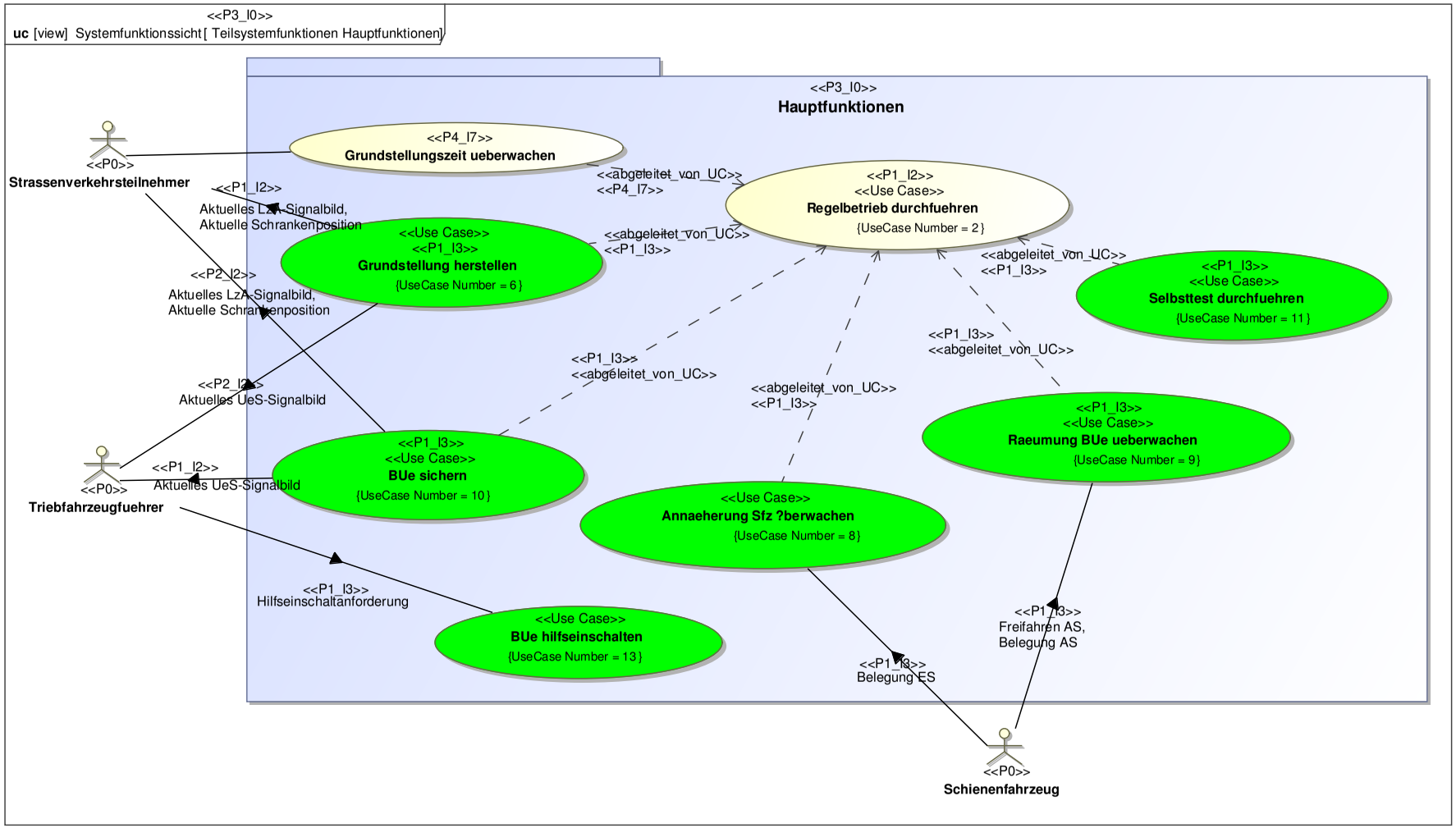


Abbildung 5.7.: Anwendungsfalldiagramm (aus dem Beispielmodell BÜ-Steuerung)

der Systemabgrenzung existieren, weil jeder Informationsfluss zwischen einem Anwendungsfall und einem Akteur automatisch auch die Systemgrenze überschreitet. Weiterhin sind die Anwendungsfälle fest dem SuB zugeordnet. Eine Information fließt damit nicht nur zwischen einem Anwendungsfall und einem Akteur, sondern automatisch auch zwischen dem SuB und einem Akteur. Daher muss der entsprechende Informationsfluss mit allen übertragenen Signalen zwingend auch in der Systemabgrenzung modelliert werden. Auch diese Konsistenzbedingungen werden in Kapitel 5.1.5 formal beschrieben. Tabelle 5.2 fasst die im Anwendungsfalldiagramm dargestellten Aspekte zusammen und zeigt die jeweils empfohlenen SysML-Beschreibungsmittel.

Inhaltselement	**SysML-Sprachelement**	**Anmerkungen**
Diagrammart	*Use Case Diagram*	
Definition der Systemfunktionen	Use cases	Die Prozessdefinition kann einen hierarchischen Baum von Anwendungsfällen erzeugen. Im endgültigen Anwendungsfalldiagramm sollte aber nur die unterste Ebene der Anwendungsfälle enthalten sein. Die einzelnen Anwendungsfälle werden durch Szenarien weiter detailliert.
Akteure	Actors	Konsistenzkriterien: Alle Akteure in der Systemumgebung müssen Beziehungen zu mindestens einem Anwendungsfall haben. Die Menge der Akteure muss der Menge der Akteure in der Systemabgrenzungssicht und der Szenariensicht entsprechen; die Systemverhaltenssicht muss mindestens eine Teilmenge der Akteure enthalten (siehe 5.1.5.2).
Darstellung der Informationsflüsse zwischen Akteur und System	Item flows	Konsistenzkriterium: Die Menge der Informationsflüsse zwischen den einzelnen Anwendungsfällen und Akteuren muss der Menge der Informationsflüsse in der Systemabgrenzungssicht zwischen SuB und Akteuren entsprechen (siehe 5.1.5.3)
Ausgetauschte Informationen	Signals	Nutzlast wird durch Attribute modelliert. Konsistenzbedingungen: Die Menge der Signale muss der Menge der Signale in der Systemabgrenzungssicht, der Szenariensicht, der Systemverhaltenssicht und der Subsystemsicht entsprechen (siehe 5.1.5.1)

Tabelle 5.2.: Modellierungselemente für die Anwendungsfallmodellierung

Mit der Systemfunktionssicht enthält das Modell somit eine Darstellung darüber,

- welche Funktionen vom SuB bereitgestellt werden sollen,
- welche Akteure welche dieser Funktionen benötigen,
- welche Informationen zwischen dem SuB und den Akteuren ausgetauscht werden sollen.

Bild 5.8 zeigt den für die Systemfunktionssicht zutreffenden Ausschnitt aus dem Struktur-Metamodell.

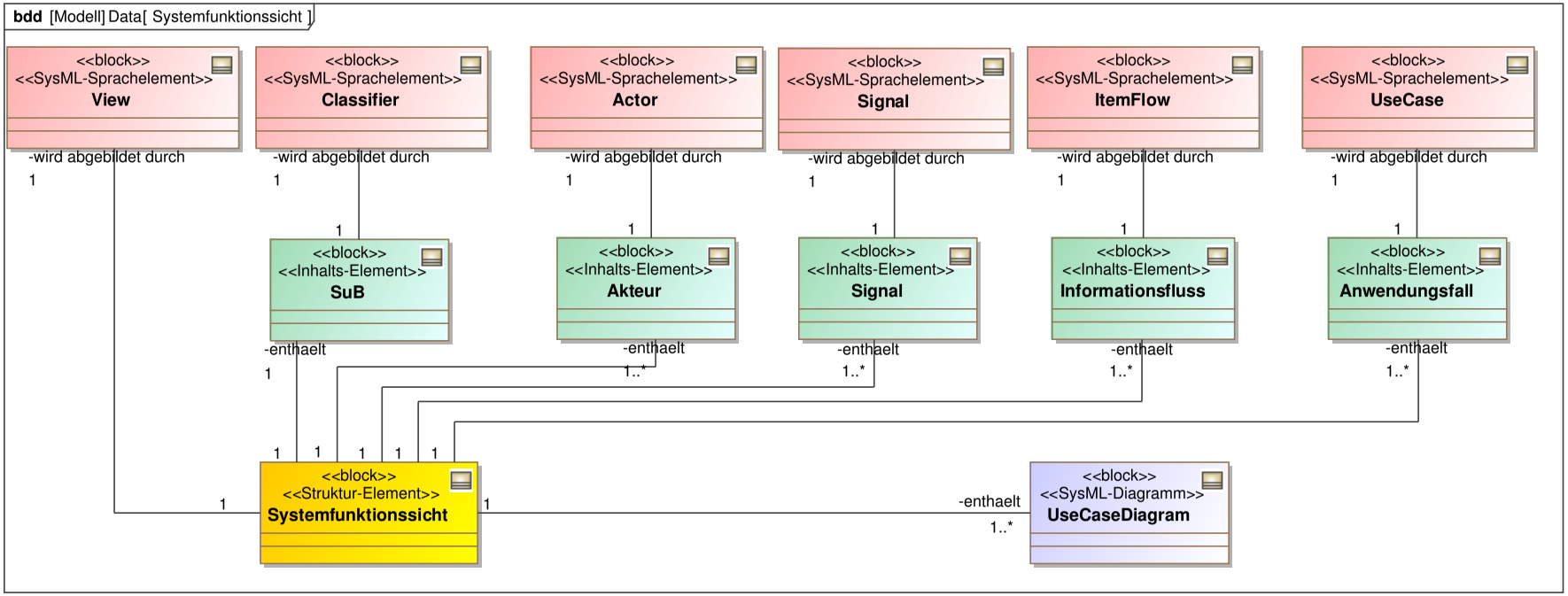


Abbildung 5.8.: Metamodell-Definition für die Systemfunktionssicht

5.1.4.3. Szenariensicht

Wie in 5.1.4.2.2 dargelegt, werden die Anwendungsfälle durch einzelne Szenarien weiter detailliert. In jedem einzelnen Szenario wird die Ausprägung des Anwendungsfalls unter einer Menge an definierten Randbedingungen beschrieben. Ändert sich eine der Randbedingungen, entsteht dadurch ein neues Szenario. Üblicherweise liegt beispielsweise für den Regelfall eines Anwendungsfalls und für alle möglichen Störungsfälle jeweils ein Szenario im Anforderungsmodell vor. Weitere Informationen zum Vorgehen bei der Szenarioerstellung finden sich in den entsprechenden Abschnitten der Prozessbeschreibung (Details zur entsprechenden Phase siehe Abschnitt 6.3.3.2).

Ziel der Szenariensicht ist eine Darstellung der Interaktionen zwischen dem SuB und den Akteuren und damit eine Konkretisierung der Informationen aus der Systemfunktionssicht. Diese enthält lediglich die Information, dass ein Akteur mit einer bestimmten Systemfunktion assoziiert ist und mit dieser Informationen austauscht. Es fehlt jedoch eine Aussage, in welcher Richtung, in welcher Reihenfolge und in welchem zeitlichen Ablauf in bestimmten Situationen die einzelnen Informationen ausgetauscht werden, sprich, wie SuB und Umgebung interagieren. Ein Szenario beschreibt diese fehlenden Aspekte und spezifiziert damit

- das Verhalten des SuB
- beispielhaft in einer bestimmten Situation
- in einer Teilmodell-bezogenen Blackbox-Sicht auf die Interaktion mit der Umgebung

Die UML stellt mit der Gruppe der Interaktionsdiagramme Beschreibungsmittel bereit, durch die genau diese Aspekte beschrieben werden können. Die SysML enthält als Vereinfachung nur das *sequence diagram* (Sequenzdiagramm), das für die Modellierung der Szenariensicht angewendet wird.

Wie der SysML-Spezifikation entnommen werden kann, entspricht die Intention der Sequenzdiagramme dabei recht genau den oben beschriebenen Anforderungen:

> „The sequence diagram describes the flow of control between actors and systems (blocks) or between parts of a system. This diagram represents the sending and receiving of messages between the interacting entities called lifelines, where time is represented along the vertical axis. The sequence diagrams can represent highly complex interactions with special constructs to represent various types of control logic, reference interactions on other sequence diagrams, and decomposition of lifelines into their constituent parts." [OMG07-1, S. 107]

Wie weiter unten jedoch noch gezeigt werden wird, sind die im letzten Satz angesprochenen komplexen Kontrollkonstrukte eher kontraproduktiv für die Lesbarkeit und Übersichtlichkeit und werden daher im Rahmen dieser Arbeit nicht verwendet.

Sequenzdiagramme an sich bestehen aus zwei wesentlichen Elementen: den *life lines* (Lebenslinien), die je ein Objekt entlang einer Zeitachse repräsentieren und den *messages* (Nachrichten), die zwischen den Objekten ausgetauscht werden. Zu beachten ist, dass die Zeitachse im Sequenzdiagramm keine Skala besitzt. So ist zwar die Reihenfolge des Nachrichtenaustauschs durch die Reihenfolge der Nachrichtenpfeile im Diagramm determiniert, jedoch lassen die Abstände der einzelnen Nachrichten im Diagramm keinen Rückschluss auf den Echtzeitablauf der Kommunikation zu. Sollen zeitliche Randbedingungen berücksichtigt werden, muss dies explizit über *timinig constraints* (Zeitbedingungen) geschehen, über die eine Zeitspanne zwischen zwei Nachrichten definiert wird. Je nach verwendetem Ausdruck ist die Vorgabe eines Minimal- oder Maximalwertes oder einer exakten Zeitspanne möglich.

Im Rahmen des Anforderungsmodells können die Sequenzdiagramme dabei als situationsbezogene Vorgabe der Interaktion zwischen SuB und den Akteuren verstanden werden. Die einzelnen Diagramme der Szenarien werden dabei nicht notwendigerweise ausschließlich händisch erstellt. Wie im entsprechenden Kapitel der Prozessbeschreibung dargelegt wird (siehe Abschnitt 6.4.5.4), bestehen die Szenarien üblicherweise aus einer Vereinigung von manuell erstellten und durch ein Testfallgenerierungswerkzeug erzeugten Sequenzdiagrammen. Hintergrund für dieses Vorgehen ist die Schaffung einer einheitlichen Testbasis, die für die Verifikation der Modellierung des Systemverhaltens erforderlich ist. Es ist somit eine permanente Konsistenz zwischen Szenarien und Systemverhalten gewährleistet, d.h. das modellierte Systemverhalten (siehe 5.1.4.4) erfüllt immer die Vorgaben aus den Szenarien. Die Szenarien dienen somit nicht nur einer Darstellung der System-Umwelt-Interaktion, sondern sind auch Testfälle sowohl für die Prozessschritte innerhalb der Modellerstellung (siehe 6.4.5.1), als auch für die spätere Abnahme des realen Systems.

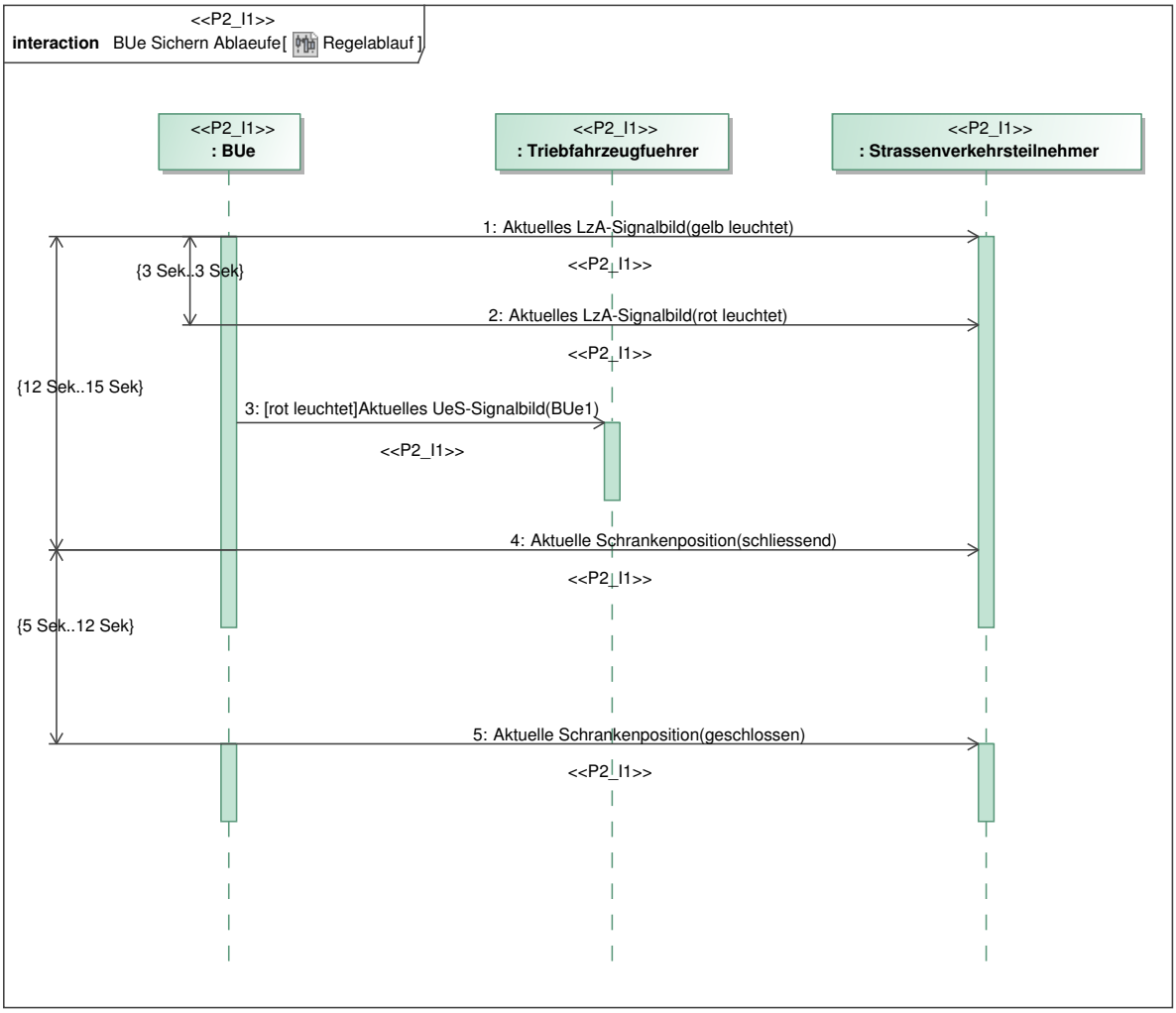


Abbildung 5.9.: Darstellung eines Szenarios in einem Sequenzdiagramm

Bild 5.9 zeigt beispielhaft ein Sequenzdiagramm zur Beschreibung eines Ablaufs. Zu erkennen sind die drei Lebenslinien, wobei die linke das SuB repräsentiert, die anderen beiden die am Szenario beteiligten Akteure, mit denen das SuB im vorliegenden Fall Informationen austauscht.

Auch die Sequenzdiagramme stellen somit eine Blackbox-Sicht auf das SuB des aktuellen Teilmodells dar. Dargestellt wird nur die nach außen sichtbare Interaktion, nicht aber die inneren Abläufe, die zur Erzielung dieser Interaktionen führen. Aus diesem Grund sollte auf die Verwendung von so genannten *Nachrichten an sich selbst* verzichtet werden, die innerhalb des SuB ablaufende Ereignisse beschreiben.

Für die Inhalte der ausgetauschten Nachrichten können auch hier wieder bestimmte Konsistenzregeln abgeleitet werden: Alle Signale, die über Nachrichten zwischen den Objekten in einem Sequenzdiagramm ausgetauscht werden, müssen auch in den Diagrammen der Systemabgrenzung und der Anwendungsfalldefinition enthalten sein. Weiterhin muss gelten, dass die Menge der unterschiedlichen Signale über alle Szenarien und alle Anwendungsfälle genau der Menge der Nachrichten entspricht, die auch in der Systemabgrenzung abgebildet ist. Weiterhin muss die Menge der Akteure in der Szenariensicht - die hier durch Lebenslinien repräsentiert werden - der Menge der Akteure in den anderen betroffenen Sichten entsprechen. Details und eine formale Spezifikation dieser Bedingungen finden sich in 5.1.5.

Wie weiter oben schon angedeutet, sind jedoch nicht alle Konstrukte für die Modellierung von Sequenzdiagrammen im Rahmen dieser Arbeit sinnvoll einsetzbar. Dies betrifft insbesondere die *operators* (Operatoren), durch die sich für bestimmte Teile von Abläufen Schleifen und Entscheidungskonstrukte aufbauen lassen. Grundelement dafür sind die so genannten *combined fragments* [OMG07-2, S. 465], bei denen es sich prinzipiell um Ausschnitte aus Sequenzdiagrammen handelt, die als Block in andere Sequenzen integriert sind und dabei bestimmten Operatoren unterworfen werden können. Es existieren nach [OMG07-2, S. 466 ff.] acht verschiedene Operatoren:

- *alternatives* - es wird zwischen verschiedenen Teilabläufen ausgewählt
- *option* - ein Teil des Ablaufs ist optional und wird nur unter bestimmten Bedingungen ausgeführt
- *break* - wie Option, nur dass danach das umgebende Szenario nicht weiter ausgeführt wird
- *parallel* - die in einzelnen combined fragments dargestellten Operationen laufen parallel ab
- *strict* und *weak sequencing* - spezifizieren den Grad der Sequenzialität zwischen verschiedenen combined fragments
- *negative* - beschreibt einen Ablauf der auf keinen Fall eintreten darf
- *critical* - Nachrichten innerhalb des *combined fragment* werden nicht durch Nachrichten anderer Fragmente überlagert

Einige dieser Fragmente widersprechen dem hier beschriebene Prinzip der strengen Szenarioorientierung und sollen deshalb nicht für die Modellierung verwendet werden. Dies betrifft insbesondere die Operatoren *alternatives*, *option* und *break*. Diese Konstrukte ermöglichen verschiedene Entscheidungspfade innerhalb eines Sequenzdiagramms. Da allerdings jedes Sequenzdiagramm genau einen möglichen Ablauf innerhalb eines Anwendungsfalls beschreiben soll, ist es sinnvoller für mehrere alternative Abläufe auch jeweils verschiedene Sequenzdiagramme zu erstellen. Dies erhöht zwar die Anzahl an erforderlichen Diagrammen, aber ebenso die Lesbarkeit und Verständlichkeit. Zudem werden die Sequenzdiagramme - wie oben angemerkt - auch als Testfälle eingesetzt. Für diese Anwendung sind einzelne „flache” Szenarien ohne Entscheidungsstrukturen erforderlich.

Die für die Szenario-Erstellung geeigneten Sprachelemente sind in Tabelle 5.3 aufgeführt.

Inhaltselement	SysML-Sprachelement	Anmerkungen
Diagrammart	*Sequence Diagram*	
Beteiligte Entitäten	Life lines	Da es sich um eine Blackboxsicht handelt, existiert eine Lebenslinie für das SuB und je eine Lebenslinie für jeden beteiligten Akteur. Konsistenzkriterium: In jedem Szenario dürfen nur diejenigen Akteure enthalten sein, die eine Beziehung zum übergeordneten Anwendungsfall besitzen. Die Menge der Akteure muss der Menge der Akteure in der Systemabgrenzungssicht entsprechen; die Systemverhaltenssicht muss mindestens eine Teilmenge der Akteure enthalten (siehe 5.1.5.2).
Nachrichten	Messages	
Ausgetauschte Informationen	Signals	Konsistenzkriterium: Die Menge der Signale zwischen SuB und Akteuren muss der Menge der Signale in der Systemabgrenzungssicht, der Systemfunktionssicht, der Systemverhaltenssicht und der Subsystemsicht entsprechen (siehe 5.1.5.1)
Kontrollstrukturen	Combined fragments	Aus Gründen der Übersichtlichkeit und Lesbarkeit und für die Verwendbarkeit als Testfälle, sollten die Operatoren alternatives, options und break nicht verwendet werden. Generell sollte ein Sequenzdiagramm immer nur genau ein Szenario repräsentieren

Tabelle 5.3.: Modellierungselemente für die Szenariensicht

Durch die Szenarien-Modellierung enthält das funktionale Modell Aussagen über

- die typischen Betriebssituationen, in denen die einzelnen Systemfunktionen angewendet werden
- die Soll-Reihenfolge der Kommunikation zwischen SuB und seiner Umgebung in den einzelnen betrieblichen Situationen und damit auch
- die Soll-Reaktion des SuB auf bestimmte Stimuli im Bezug auf die einzelnen Szenarien

Die semiformale Spezifikation der in der Szenariensicht enthaltenen Modell- und Sprachelemente kann dem entsprechenden Ausschnitt aus dem Struktur-Metamodell in Bild 5.10 entnommen werden.

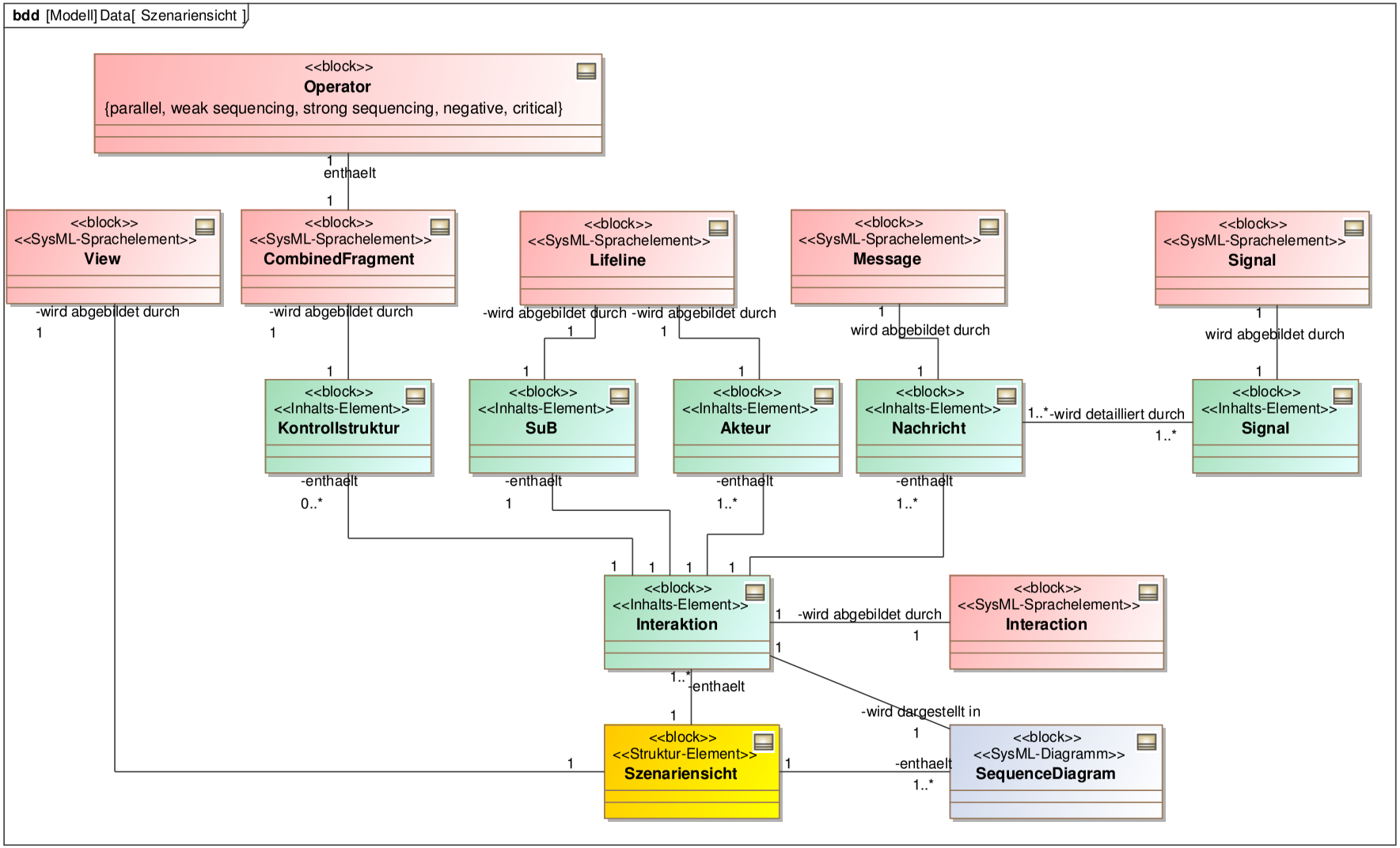


Abbildung 5.10.: Metamodell-Definition der Szenariensicht

5.1.4.4. Systemverhaltenssicht

Mit den bisher beschriebenen Sichten können die folgenden Sachverhalte im funktionalen Modell dargestellt werden:

(a) wie sich das System von seiner Umgebung abgrenzt und mit welchen Elementen seiner Umgebung es kommuniziert

(b) welche Systemfunktionen es seiner Umgebung bereitstellt

(c) wie diese Systemfunktionen in bestimmten Situationen mit der Umgebung interagieren sollen

Diese Elemente sind jedoch noch nicht hinreichend für ein ausführbares Modell, das die Grundlage für die Umsetzung der in Abschnitt 4 formulierten Ziele ist: Nur ausführbare Modelle sind test- und verifizierbar, wobei eine Verhaltensspezifikation - wie im nachfolgenden Unterkapitel gezeigt werden wird - wesentliche Voraussetzung für die Ausführbarkeit ist. Das funktionale Modell muss daher zwingend um eine Modellierung des Verhaltens ergänzt werden.

Im nachfolgenden Unterabschnitt 5.1.4.4.1 wird zunächst der Begriff der Ausführbarkeit für den Rahmen dieser Arbeit definiert und anschließend die Möglichkeiten beschrieben, die sich daraus ergeben. Anschließend wird in Unterabschnitt 5.1.4.4.2 dargelegt, welches Beschreibungsmittel für die Verhaltensmodellierung von Anforderungsmodellen geeignet erscheint. Verschiedene Aspekte der Verhaltensmodellierung werden dann in den Unterkapiteln 5.1.4.4.3 bis 5.1.4.4.6 beschrieben.

5.1.4.4.1. Möglichkeiten ausführbarer Modelle

Wesentliches Ziel einer Verhaltensspezifikation ist die Schaffung eines ausführbaren Anforderungsmodells, das die Grundlage für die Verhaltensvisualisierung, die Durchführung von Tests und Erzeugung von Testfällen und die formale Verifikation ist. Bevor im Folgenden die genannten Möglichkeiten einzeln vorgestellt werden, soll zunächst der Begriff *ausführbar* für die weitere Verwendung in dieser Arbeit definiert werden: Ein Modell ist dann ausführbar, wenn sich die Modellinformationen in eine entsprechende Zielsprache (wie C++, Java) übersetzen lassen und sich der erzeugte Quellcode in ein ausführbares Programm kompilieren lässt. Dazu sind im vorliegenden Fall dieser Arbeit[1] die folgenden Vorbedingungen zu erfüllen:

- Das Verhalten des Modells muss deterministisch sein, wobei - analog zu einer deterministischen Turing-Maschine - zu jedem möglichen Zustand des Systems der Folgezustand eindeutig definiert sein muss [WES00]
- Die Struktur des Systems und seiner Umgebung muss durch Klassendiagramme (UML) oder Blockdefinitionsdiagramme (SysML) modelliert worden sein
- Das Verhalten des Systems muss mit entsprechenden Modellierungselementen der UML/-SysML (Zustandsmaschine, Aktivitätsdiagramm) abgebildet worden sein

Bei Verwendung einer entsprechenden Modellierungsumgebung (im Rahmen dieser Arbeit: IBM/telelogic Rhapsody) lässt sich unter Berücksichtigung dieser Vorbedingungen das Modell kompilieren und ausführen. Das Verhalten des ausführbaren Modells wird in Form animierter Zustands- und Aktivitätsdiagramme angezeigt und kann intuitiv nachvollzogen werden. Durch die Rückmeldung lässt sich auffälliges Fehlverhalten bereits während der Erstellung der Anforderungsspezifikation erkennen. Zu beachten ist dabei, dass zurzeit noch keine durchgängig

[1] Dies meint die gegebenen Randbedingungen für die Modellierung, wie: Modellierungssprache (UML/SysML), Modellierungswerkzeug und verwendete Zielsprache (C++).

formal definierte Semantik für die UML bzw. SysML vorhanden ist. Die Übersetzung des Modells in ausführbaren Code erfolgt daher auf Grundlage semantischer Definitionen, die jeder Werkzeughersteller nach eigenem Ermessen erstellt. Gleiche Modelle können somit in unterschiedlichen Werkzeugen zu unterschiedlichem ausführbaren Code führen. Die Schaffung einer einheitlichen semantischen Definition der UML-Sprachelemente (die auch in der SysML weiterverwendet werden könnte) ist jedoch Gegenstand verschiedener Forschungsvorhaben, beispielsweise des *UML Sematics Project*. Eine Darstellung des aktuellen Arbeitsstandes findet sich beispielsweise in [BCD07].

Eine stringentere Methode zur Überprüfung des Modellverhaltens ist das systematische Anwenden von Testfällen. Dabei wird das zu untersuchende System von seiner Umgebung frei geschnitten und die Umwelt durch die Testumgebung simuliert. Bei der Testausführung versorgt diese den Testling mit bestimmten Stimuli und zeichnet die Reaktionen des Systems beispielsweise in Form von Sequenzdiagrammen auf. Das Testprotokoll kann dann mit dem Soll-Verhalten verglichen werden, wodurch sich Abweichungen übersichtlich erkennen lassen. Auf die allgemeinen Möglichkeiten und Grenzen des Testens mit Testfällen und die konkrete Umsetzung im Rahmen dieser Arbeit wird in Abschnitt 6.4.5 eingegangen.

Eine nochmals strengere Möglichkeit zur Überprüfung des Systemverhaltens stellt die Anwendung des *Model Checkings* (Modellprüfung) dar. Dabei handelt es sich um eine Technik zur Prüfung eines Systems aus einem endlichen Automaten [CGP00, S. 3] gegen eine formale Spezifikation. Ergebnis dieser Prüfung ist eine ja/nein-Aussage, ob das Verhalten des Systems die Spezifikation erfüllt oder verletzt. Auch diese Verifikationstechnik kann mit bestimmten Einschränkungen auf ausführbare UML/SysML-Modelle angewendet werden, was in Abschnitt 6.4.6 beschrieben wird.

Zu beachten ist, dass zwischen der Systemverhaltenssicht und der Szenariensicht eine inhaltliche Abhängigkeit besteht. Die einzelnen Szenarien stellen dabei exemplarische, kontextabhängige Protokolle des Systemverhaltens dar, das durch die Verhaltensmodellierung bestimmt wird. Da die Szenariensicht und die Verhaltenssicht jedoch zunächst unabhängig voneinander erstellt werden, müssen Mechanismen vorgesehen werden, die beide Sichten inhaltlich aneinander koppeln. Dies geschieht durch Integration von Testausführungs- und Testerzeugungstechniken, durch die ein bidirektionaler Abgleich zwischen Szenariensicht und Verhaltenssicht möglich wird. Dadurch kann sowohl überprüft werden, ob das Verhaltensmodell des SuB die als Soll-Vorgabe dienenden Szenarien erfüllt, als auch, ob tatsächlich alle zur Beschreibung des gewünschten Systemverhaltens erforderlichen Interaktionsszenarien ermittelt wurden. Diese wechselseitige Abhängigkeit ist Grundlage der im Prozess verfolgten internen Teststrategie. Details dazu finden sich in Abschnitt 6.4.5.

5.1.4.4.2. Diskussion des Beschreibungsmittels

Für die Verhaltensmodellierung bieten sich zwei Beschreibungsmittel an, die fast unverändert aus der UML 2 in die SysML übernommen wurden [OMG07-1, S. 85]: Die Modellierung durch *State Machines* (Zustandsmaschinen) oder die Modellierung mit *Activity Diagrams* (Aktivitätsdiagrammen). Im Bezug auf die Ausführbarkeit sind beide Beschreibungsmittel als gleichwertig anzusehen, da sowohl durch Zustandsmaschinen, als auch durch Aktivitätsdiagramme mit geeigneten Werkzeugen ausführbare Modelle im Sinne der Definition in 5.1.4.4.1 erstellt werden können.

Unterschiedlich ist jedoch der Reifegrad der Formalität beider Beschreibungsmittel. Die Zustandsmaschinen der UML/SysML basieren auf den Zustandsdiagrammen von Harel [HAR87] [OMG07-2, S. 519], für die eine formale Ausführungssemantik abgeleitet werden kann. Dies

hat beispielsweise Arabestani in [ARA05] durchgeführt. Die Aktivitätsdiagramme verwenden eine Mischung aus dem Tokentransport der Petri-Netze [OMG07-2, S. 295] und einer Datenflussnotation (*ObjectFlows*, [OMG07-2, S. 386]). Dabei hinkt die wissenschaftliche Aufarbeitung der Semantik von Aktivitätsdiagrammen hinter der beschriebenen pragmatisch-technischen Umsetzung durch die Werkzeughersteller und dem erreichten Arbeitsstand bei den Zustandsautomaten hinterher. Neben dem generellen Ansatz des UML Semantics Project werden in diversen Forschungsarbeiten zurzeit dedizierte Ansätze für eine formale Definition von Aktivitätsdiagrammen beschrieben. Dabei werden verschiedene Strategien verfolgt: So wird beispielsweise versucht, die Aktivitätsdiagramme in Petri-Netze zu transformieren [STÖ04]. Andere Ansätze basieren auf der Verwendung von kleinen, virtuellen Maschinen zum Ausführen der einzelnen Aktivitäten, wobei hierbei meist nur eine Teilmenge der Sprachelemente von Aktivitätsdiagrammen implementiert wird [VIK05]. Wieder andere Autoren versuchen Aktivitätsdiagramme durch Zustandsmaschinen auszudrücken, wobei oftmals aber nur UML 1.5-konforme Sprachkonstrukte verarbeitet werden können. Eshuis beschreibt in [ESW01] eine formale Semantik für Aktivitätsdiagramme und zeigt in [ESH06], dass sich aus Aktivitätsdiagrammen Eingangsdaten für den symbolischen Model-Checker NuSMV erzeugen lassen, womit auch das in Aktivitätsdiagrammen beschriebene Verhalten formal verifizierbar wird.

Eine idealisierte Zielvorstellung - als Fortführung der genannten Arbeiten - wäre eine formal fundierte Semantik für Aktivitätsdiagramme analog [ARA05]. Die Semantik müsste die direkte Ausführbarkeit von Aktivitätsdiagrammen sowie deren Einsatz für Testfallausführung, Testfallgenerierung und formale Verifikation ermöglichen. Jedoch erscheint es für den Bereich der Anforderungsmodellierung vertretbar, bereits jetzt die Vorteile der in zahlreichen Werkzeugen implementierten Codeerzeugung für Aktivitätsdiagramme auch ohne strikte Semantikdefinition zu nutzen. Zuvor sollte jedoch in gesonderten Arbeiten das aus unzureichend formal definierter Codeerzeugung entstehende Risiko analysiert werden.

Neben diesen Differenzen im Grad des Formalismus unterscheiden sich Zustandsmaschinen und Aktivitätsdiagramme auch aus Sicht der Modellierung und damit auch die Anwendungsschwerpunkte beider Beschreibungsmittel. Die folgende Auflistung differenziert die wichtigsten Unterschiede aus einer pragmatischen Sicht auf die Modellierung:

(a) Bei Zustandsautomaten liegt der Schwerpunkt auf den Zuständen, in denen sich die zugehörigen Komponenten (Klassen, Blöcke, Systeme, etc.) befinden sowie auf den Transitionen (optional mit Ereignis, Bedingung und auszuführender Aktion), die Zustandsänderungen darstellen. Daher eignen sich Zustandsautomaten gut für die Beschreibung von Systemen, für die sich diskrete Zustände ermitteln lassen. Für eine weiterführende, detaillierte Beschreibung der Möglichkeiten und der Syntax von Zustandsmaschinen sei auf [WEI06, S. 272 ff.] und [OMG07-1, S. 519, ff.] verwiesen.

(b) Aktivitätsdiagramme sind im Gegensatz zu den Zustandsmaschinen geeigneter für ablauforientierte Beschreibungen, ähnlich z.B. Flussdiagrammen. Einzelne Aktivitäten repräsentieren nicht zwangsläufig einen bestimmten Systemzustand und die Übergänge zwischen zwei Aktivitäten sind nicht zwangsweise an das Auftreten von Ereignissen gebunden. Außerdem können Aktivitäten in der SysML-Version von Aktivitätsdiagrammen auch kontinuierlich ablaufen, wenn beispielsweise Stoff- oder Informationsflüsse (Luft, Wasser, Datenströme, etc.) modelliert werden sollen. Weiterhin kann in Aktivitätsdiagrammen durch das Konstrukt der Objektflüsse auch der Austausch von Daten zwischen den einzelnen Aktivitäten explizit dargestellt werden. Weitere Informationen über die Aktivitätsdiagramme finden sich in [WEI06, S. 251 ff.], in [OMG07-1, S. 85 ff.] und in [OMG07-2, S. 295 ff.].

Während der Modellierung im Rahmen dieser Arbeit hat sich gezeigt, dass Aktivitätsdiagramme wegen der in (b) genannten Eigenschaften für anforderungsbezogene Problemstellungen übli-

cherweise das mächtigere und universellere Werkzeug zur Verhaltensmodellierung sind. Zudem fällt den meisten Domänenexperten offenbar die flussorientierte Beschreibung mit Aktivitätsdiagrammen leichter, als eine Beschreibung mit Zustandsmaschinen. Dies zeigt sich beispielsweise daran, dass zur Modellierung von Geschäftsprozessen - einer zur Anforderungsmodellierung ähnlichen Fachdisziplin mit einem vergleichbaren Abstraktionsgrad - zahlreiche Konzepte auf Basis der Aktivitätsdiagramme entwickelt wurden, allen voran die ebenfalls von der OMG entwickelte BPMN [OMG06-2]. Dies ist offenbar ein Folge davon, dass Beschreibungsmittel nur dann in der Praxis akzeptiert werden, wenn sie den Anforderungen und Vorstellungen der Benutzer entgegenkommen. Aus diesem Grund sollte bei der Wahl der Beschreibungsmittel für den Produktiv-Betrieb in erster Linie die Erfüllung eben dieser Nutzeranforderungen berücksichtigt werden.

Jedoch kann die große Vielseitigkeit und Mächtigkeit der Aktivitätsdiagramme insbesondere wenig erfahrenen Modellierern den Einstieg in die Verhaltensmodellierung erschweren. Daher werden in 5.1.4.4.5 in Form eines Leitfadens bewährte Modellierungsmuster für häufig auftretende Problemstellungen zusammengestellt.

Von der generellen Bevorzugung von Aktivitätsdiagrammen in dieser Arbeit bleibt unbeeinflusst, dass für bestimmte Problemstellungen auch Zustandsmaschinen das geeignetere Beschreibungsmittel sein können. Als Prämisse für die Verhaltensmodellierung lässt sich anhand der oben getroffenen Aussagen demnach folgendermaßen zusammenfassen: Standardmäßig sollten Aktivitätsdiagramme zur Verhaltensmodellierung verwendet werden - es sei denn, auf Grund der jeweiligen Problemstellung ist von vornherein klar, dass Zustandsmaschinen besser zur Modellierung geeignet sind.

5.1.4.4.3. Integration des Systemverhaltens ins Anforderungsmodell

Die Modellierung des Systemverhaltens erfolgt - entsprechend der Definition der Phase 02 der Prozessbeschreibung (siehe Abschnitt 6.3.3) - basierend auf den Systemfunktionen. Bei deren Ableitung kann, je nach Komplexität des beschriebenen Systems, bereits ein grobes Verhaltensmodell entstanden sein (siehe Abschnitt 6.4.2), das auf der Beschreibung von zunächst sehr allgemeinen Anwendungsfällen durch Aktivitätsdiagramme beruht. Ein Vorhandensein solcher Strukturen wird dabei für die folgenden Ausführungen angenommen. Die hier beschriebene Verhaltensmodellierung stellt dann einen weiteren Verfeinerungsschritt dar, dessen Einordnung in die Struktur des gesamten Anforderungsmodells dem Bild 5.11 entnommen werden kann.

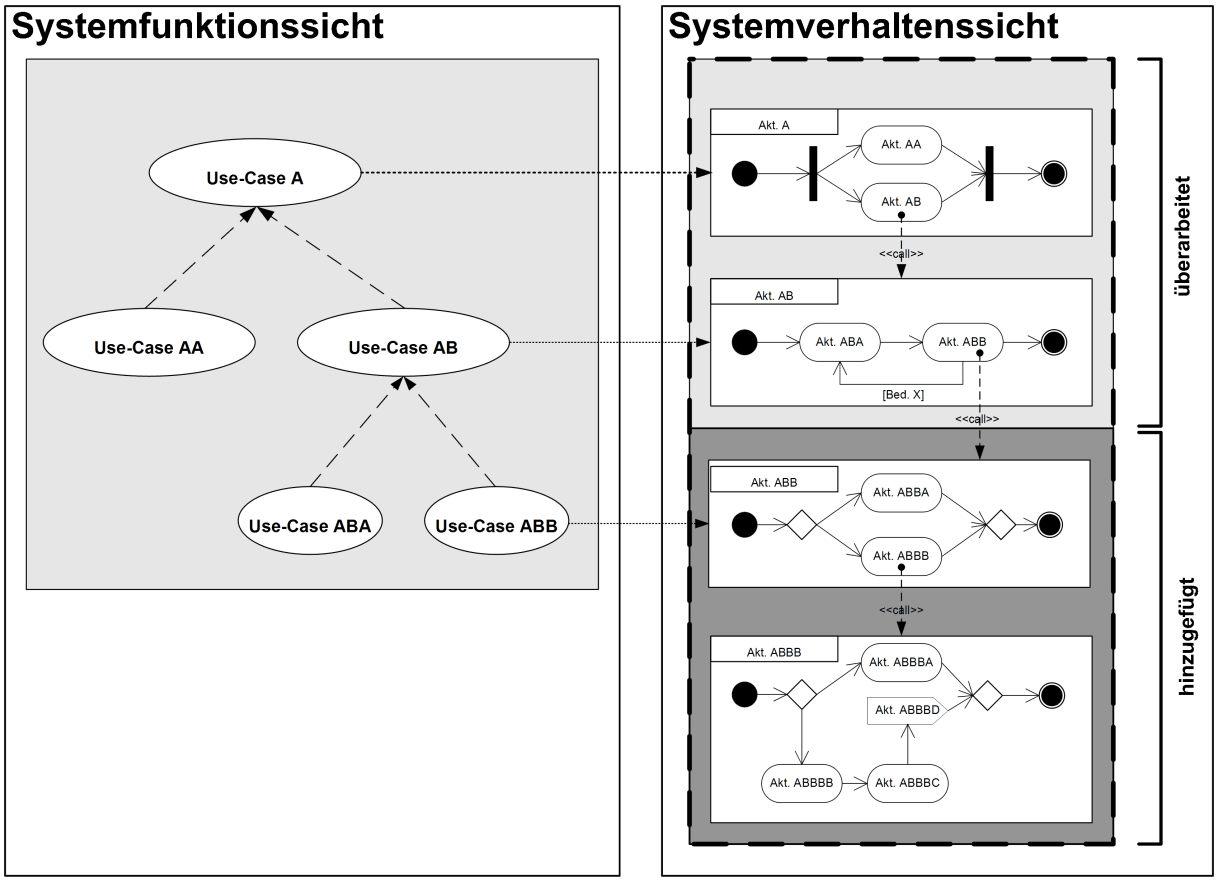


Abbildung 5.11.: Einordnung des Systemverhaltens in das funktionale Modell

Die hellgrau hinterlegten Modellelemente entstammen dabei den Prozessschritten zur Funktionsableitung. Die dort erstellten Verhaltensdiagramme sind jedoch für die oben beschriebene Ausführbarkeit der Anforderungsspezifikation üblicherweise zu wenig detailliert. Daher müssen im weiteren Prozessablauf zum einen die bereits vorhandenen Diagramme überarbeitet[2] und zum anderen weitere Verfeinerungsschritte hinzugefügt werden. Die neu hinzugefügten Modellelemente gliedern sich dabei in den im Bild 5.11 dunkelgrau hinterlegten Bereich in das Anforderungsmodell ein. Auf Grund des Prozessablaufs ist dabei immer sichergestellt, dass die neu hinzugefügten Verhaltenskonstrukte jeweils einzelnen Systemfunktionen zugeordnet sind.

Das Gesamtverhalten des Systems wird letztendlich durch die folgenden Elemente determiniert:

- verfeinerte Verhaltensdiagramme aus der Prozessphase „Spezifikation der Systemfunktionen” (hellgrau hinterlegt, Erstellung siehe 6.4.2)
- neue Diagramme aus der Prozessphase „Spezifikation des Systemverhaltens” (dunkelgrau hinterlegt, Erstellung siehe 6.4.4)

Alle Verhaltensdiagramme zusammen bilden eine baumähnliche Struktur, die dem Konzept einer schrittweisen Verfeinerung des Systemverhaltens entspricht. Ein geeignetes Modellierungswerkzeug vorausgesetzt, lässt sich durch diese Struktur einfach hindurchnavigieren, um das Systemverhalten auf unterschiedlichen Abstraktionsniveaus einzusehen.

[2] Dabei muss durch den Prozess sichergestellt werden, dass die Verfeinerungen funktionserhaltend sind, also durch die Überarbeitung keine Veränderung des modellierten Verhaltens eintritt.

5.1.4.4.4. Aktionssprachen

In den Diagrammen der Systemverhaltenssicht kann ein Teil der zur Ausführbarkeit nötigen Informationen direkt durch die grafischen Sprachelemente der UML/SysML ausgedrückt werden. Vergleicht man Aktivitäts- und Zustandsdiagramme hinsichtlich dieser Eigenschaft, zeigt sich, dass Aktivitätsdiagramme durch das Konzept der *actions* (Aktionen) [OMG07-2, S. 219 ff.] dabei eine höhere semantische Abdeckung aufweisen. Bei Aktionen handelt es sich um vordefinierte Verhaltenselemente für bestimmte Routineaufgaben, beispielsweise existieren Aktionen für das Lesen und Schreiben von Attributen, das Senden und Empfangen von Nachrichten und für Entscheidungs- und Verzweigungskonstrukte. Die Verwendbarkeit dieser Sprachelemente für ausführbare Modelle ist allerdings stark davon abhängig, ob und ggf. für wie viele der Standard-Aktionen das gewählte Modellierungswerkzeug eine Code-Erzeugung unterstützt. So implementiert das verwendete IBM/telelogic Rhapsody nur einen Bruchteil der in UML und SysML definierten Aktionen.

Existieren für bestimmte Teile des Zielverhaltens keine grafischen UML/SysML-Sprachkonstrukte, müssen die nötigen Informationen anderweitig ausgedrückt werden. In vielen Modellierungswerkzeugen geschieht dies durch Fragmente der gewählten Ausführungs-Zielsprache, also z.B. durch entsprechenden C++-Code, der bestimmten Modellelementen (beispielsweise als *guard* einer Transition) zugeordnet wird.

Dies ist in mehrfacher Hinsicht ungünstig:

- Das Modell wird somit auf eine Ausführungs-Zielsprache festgelegt. Möchte man das gleiche Modell in eine andere Sprache - wie z.B. Java - übersetzen, müssen alle händisch eingefügten Statements der einen Zielsprache aufwändig durch die entsprechenden Äquivalente der anderen Sprache ersetzt werden. Damit wird nicht nur der eigentlich abstrakte, grafische Modellierungsansatz der UML/SysML durchbrochen, sondern auch eine Möglichkeit für zusätzliche Modellierungsfehler geschaffen.
- Der Modellierer wird dazu gezwungen, sich Kenntnisse der jeweiligen Zielsprache anzueignen, damit er die erforderlichen Sprachausdrücke erstellen kann. Dies mag zwar im Bereich der Softwareentwicklung vertretbar sein, ist aber im Bereich der Anforderungserstellung für Systeme besonders ungünstig. Der Modellbearbeiter ist dort meist mehr mit der zugehörigen Fachdomäne als mit Programmiersprachen vertraut.
- Selbst bei prinzipiellen Kenntnissen über die Zielsprache verwenden viele Modellierungswerkzeuge umfangreiche Frameworks, um den UML/SysML-Sprachumfang abzubilden. Daher ist nicht nur Wissen über die Zielsprache an sich erforderlich, sondern auch über deren konkrete Verwendung in der werkzeugspezifischen Systemumgebung.

Dieses grundsätzliche Problem der UML/SysML ist in dieser Form durchaus bekannt, so dass verschiedene Ansätze zur Lösung entwickelt wurden. Diese lassen sich grob in zwei Gruppen einteilen: Zum einen in Ansätze zur Entwicklung einer zielsprachenunabhängigen Aktionssprache und zum anderen in die Einführung des Aktionskonzeptes in der UML 2.0. Zur ersten Gruppe zählen Ideen zur Erweiterung bestehender Sprachen, beispielsweise in Form einer Erweiterung der *objects constraint language* (OCL, [OMG06-1]) zur OCL4X [JZM07], oder auch komplett neu entwickelte Sprachen [MTA98, DOU01]. Allerdings wird durch dieses Vorgehen lediglich die Problematik der Abhängigkeit von einer spezifischen Zielsprache und zur Werkzeugumgebung gelöst. Der Modellbearbeiter muss dennoch die jeweilige Spezifikationssprache zusätzlich zur UML-Notation beherrschen.

Daher erscheint es aus Sicht der Anwendungsdomain sinnvoller, wo immer es möglich ist, das UML/SysML-Aktionskonzept zu verwenden, um möglichst wenig Berührungspunkte zur darunter liegenden Zielsprache und zum Werkzeug-Framework zu erzeugen. Im Rahmen dieser Arbeit

wird - sofern möglich - eben diese Strategie verfolgt und versucht, sämtliche zur Ausführung nötigen Statements durch entsprechende UML-Aktionen abzubilden. Ist dies nicht möglich, so werden die fehlenden Statements durch C++-Code ergänzt, der an das Rhapsody-Framework angepasst ist.

5.1.4.4.5. Modellierung der Systemverhaltenssicht in der SysML

Entgegen dem Vorgehen in den anderen Kapiteln soll der Fokus dieses Unterabschnittes nicht so sehr auf einer Diskussion der einzelnen Notationselemente liegen, sondern eher auf der Beschreibung einzelner Modellierungsmuster (also von Gruppen einzelner Notationselemente) für wichtige, immer wiederkehrende Problemstellungen. Dies liegt zum einen darin begründet, dass sowohl Aktivitäts-, als auch Zustandsdiagramme eine Vielzahl von Notationselementen aufweisen. Je nach Aufgabenstellung ist eine andere Untermenge dieser Notationselemente sinnvoll, weswegen keine einfache Unterscheidung in geeignete und ungeeignete Sprachelemente möglich ist. Daher enthält die Sprachdefinition der SysML(A) auch alle verhaltensspezifizierende Elemente der UML bzw. SysML. Zum anderen ist es gerade auf Grund dieser Komplexität sinnvoll, Hinweise zu geben, welche Entwurfsmuster sich in der Modellierungspraxis bewährt haben und daher bei der Erstellung der Verhaltensmodelle bevorzugt werden sollten. Dabei wird berücksichtigt, dass entsprechend den Aussagen aus dem vorherigen Kapitel möglichst viele der UML/SysML-Aktionen verwendet werden, um den Einsatz einer Aktionssprache möglichst zu vermeiden.

Senden von Signalen

Eine besondere Charakteristik reaktiver Systeme ist die Kommunikation zwischen Systembestandteilen untereinander und von Systembestandteilen und der Systemumgebung. Wie bereits bei der Beschreibung der Systemabgrenzung dargelegt, erfolgt die ereignisorientierte Kommunikation dabei über Ports und Signale. Für jeden Signalempfang und jede Aussendung eines Signals lässt sich genau ein diskreter Zeitpunkt bestimmen.

Das Aussenden von Signalen aus einem Verhaltensmodell heraus ist typischerweise dann erforderlich, wenn in einem Umgebungsobjekt oder einer Systemkomponente eine Aktion ausgelöst werden soll. Anwendungsfälle im Eisenbahnwesen für dieses Modellierungskonstrukt könnten sein:

- Senden des Einschaltanstoßes vom Einschaltkontakt an die Bahnübergangssicherungsanlage
- Auslösen einer Zwangsbremsung durch die Zugbeeinflussung mittels Senden eines entsprechenden Signals an die Bremseinwirkbaugruppe
- Auch möglich ist die Darstellung der Zustandsänderung von Komponenten. So ließe sich beispielsweise die Fahrtstellung eines Hauptsignalbildes durch das Senden eines (UML-Signals) vom Hauptsignal-Objekt zum Akteur des Triebfahrzeugführers mit dem Parameter „Aktuelles Signalbild = Ks1" modellieren.

Nicht geeignet ist das hier beschriebene Modellierungsmuster jedoch für das kontinuierliche Senden und Empfangen von Daten- oder Stoffströmen. Dies sollte über Flowports und eine entsprechende Aktivitätsmodellierung mit Objektknoten oder *SendObjectActions* geschehen.

Modellierungselement für das Versenden von Signalen ist die in UML vordefinierte Aktion *SendSignalAction* [OMG07-2, S. 284]. Sie erwartet als Argumente das zu versendende Signal und das Zielobjekt. Die spezielle pfeilförmige Notation dieser Aktion gibt einen Hinweis auf das Verhalten der Aktion, wie Bild 5.12 zeigt.

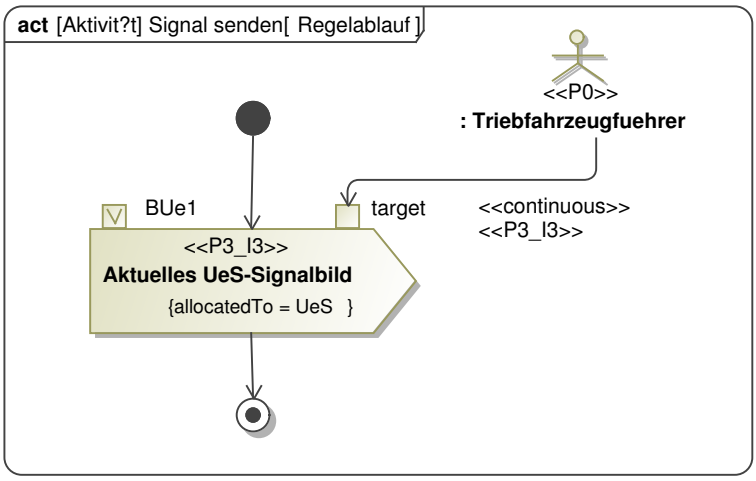


Abbildung 5.12.: Beispiel für eine SendSignalAction

In diesem Beispiel wird das Signal „Aktuelles UeS-Signalbild" an das Umgebungsobjekt „Triebfahrzeugführer" geschickt und zusätzlich mit „BÜ1" parametrisiert.

Auf das gesendete Signal lassen sich beliebige „Nutzlasten" - beispielsweise komplexe Datenstrukturen zur Parametrisierung - aufsatteln, da der Typ *signal* zur Menge der *classifier* gehört und daher mit beliebigen Parametern versehen werden kann. Konkrete Parameter können über *Value-Pins* modelliert werden, wie z.B. die Zuweisung des Wertes BÜ1 in Bild 5.12.

Ein Spezialfall der Nachrichtenübermittlung ist die *BroadcastSignalAction* [OMG07-2, S. 244 ff.], die ebenfalls ein *signal* versendet. Allerdings wird das spezifizierte Signal nicht nur an einen bestimmten Empfänger gesendet, sondern (implementierungsabhängig) gleichzeitig an einige oder alle Objekte im Modell. Damit könnte beispielsweise die Übertragung eines LZB-Nothaltauftrages an alle Züge in einem bestimmten Streckenbereich modelliert werden. Zur Unterscheidung der *SendSignalAction* und der *BroadcastSignalAction* hat letztere die normale Notation einer Aktion und den Stereotyp „*BroadcastSignal*".

Auch für das Senden von Signalen lassen sich Konsistenzbedingungen formulieren, die bei der Modellerstellung eingehalten werden müssen. So muss jedes zu verschickende Signal in der Spezifikation der Systemabgrenzung, der Systemfunktionen und der Betriebsszenarien als ausgehendes Kommunikationsfragment enthalten sein. Umgekehrt muss für jede in diesen Modellelementen als ausgehend definierte Nachricht auch eine *SendSignalAction* existieren, die diese Nachricht tatsächlich versendet.

Senden von Objekten

Sollen keine diskreten Signale, sondern komplexere Entitäten zwischen Systembestandteilen oder mit der Umgebung ausgetauscht werden, bieten sich dabei zwei verschiedene Modellierungsstrategien an. Die eine nutzt die vordefinierte *SendObjectAction* [OMG07-2, S. 404 ff.] und eignet sich für das zeitdiskrete Versenden komplexer Objekte. Möchte man hingegen einen kontinuierlichen Stoff- oder Informationsfluss darstellen, so muss auf eine implizite Modellierung mit entsprechenden Objektknoten zurückgegriffen werden. Beide Konzepte werden im Folgenden vorgestellt.

Die vordefinierte Aktion *SendObjectAction* erwartet zwei Parameter, die der Aktion über Eingabe-Pins bereitgestellt werden müssen: Zum einen ein Objektfluss vom zum verschickenden Objekt

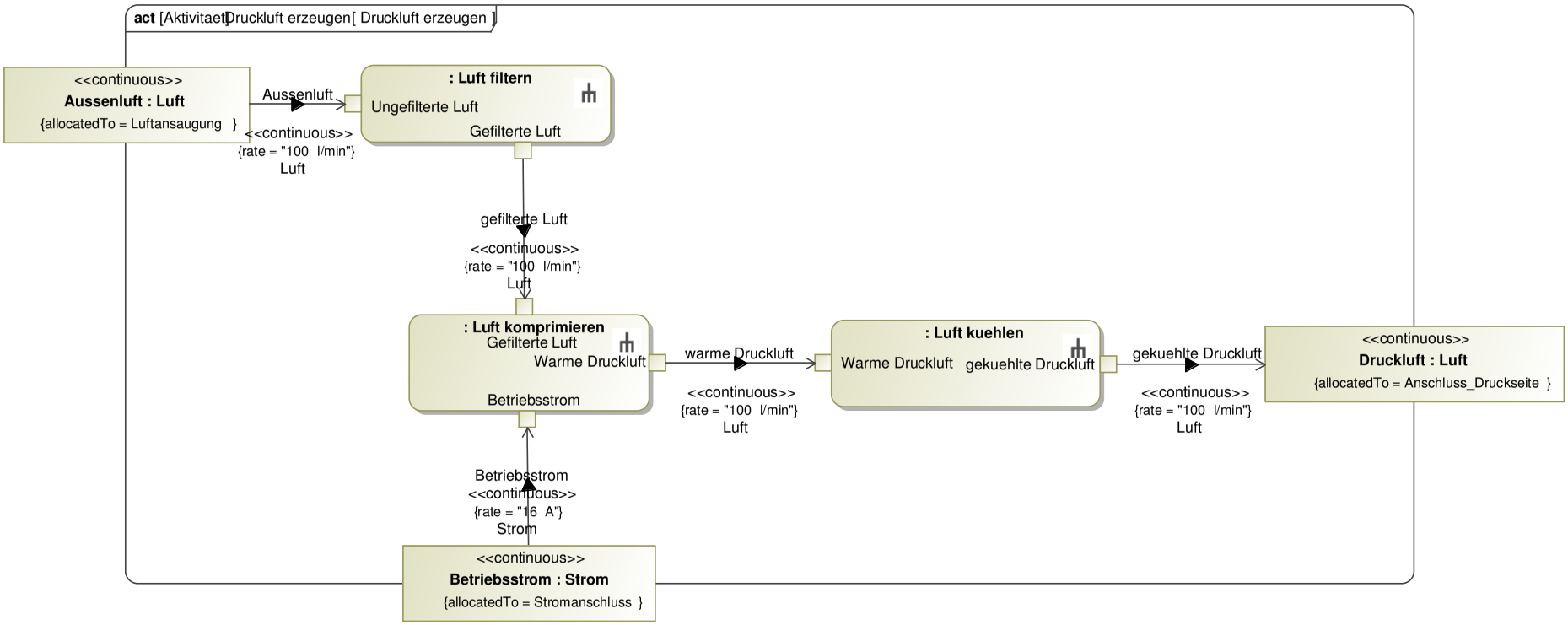


Abbildung 5.13.: Implizite Modellierung von Objektflüssen (Aktivität)

in den *request*-Pin und zum anderen ein Objektfluss in den *target*-Pin, wodurch das Zielobjekt definiert wird. Zu Beachten ist dabei, dass das Senden des Objekts genau dann einmal ausgelöst wird, wenn der Kontrollfluss die *SendObjectAction* erreicht.

Soll jedoch im Modell dargestellt werden, dass bestimmte Informationen und Stoffflüsse kontinuierlich ausgegeben werden, eignet sich dieses Vorgehen nicht. Für diesen Fall sollte eine Modellierung über *ObjectNodes* (Objektknoten) oder *ActivityParameters* (Aktivitätsparameter) realisiert werden. Zusammen mit der Möglichkeit, einzelnen Aktionen ein Architekturartefakt zuzuweisen, kann damit implizit der Fluss von Objekten auf Ports (und damit weiter auf die mit dem Port verbundene Entität) beschrieben werden. Folgendes Beispielbild 5.13 zeigt dieses Modellierungsmuster unter Nutzung von Aktivitätsparametern.

Relevant ist hierbei der rechts erkennbare Aktivitätsparameter „Druckluft" vom Typ „Luft". Dieser erhält von der Aktivität „Luft kühlen" einen kontinuierlichen Strom von Druckluft-Objekten. Die Verknüpfung zur Architektur und damit zum Ziel des Druckluftstroms erfolgt durch die Allokation des Parameters „Druckluft" auf die Komponente „Anschluss_Druckseite". Betrachtet man die dem Beispiel zugrunde liegende Systemabgrenzung (siehe Bild 5.14), erkennt man, dass es sich bei diesem Architekturartefakt um einen Objektflussport an der Systemgrenze handelt. Aus beiden Diagrammen zusammen kann somit die Information entnommen werden, dass die in der Aktivität erzeugte „Druckluft" über den Port „Anschluss_Druckseite" an den Akteur „Luftbehälter" in einem kontinuierlichen Strom abgegeben wird.

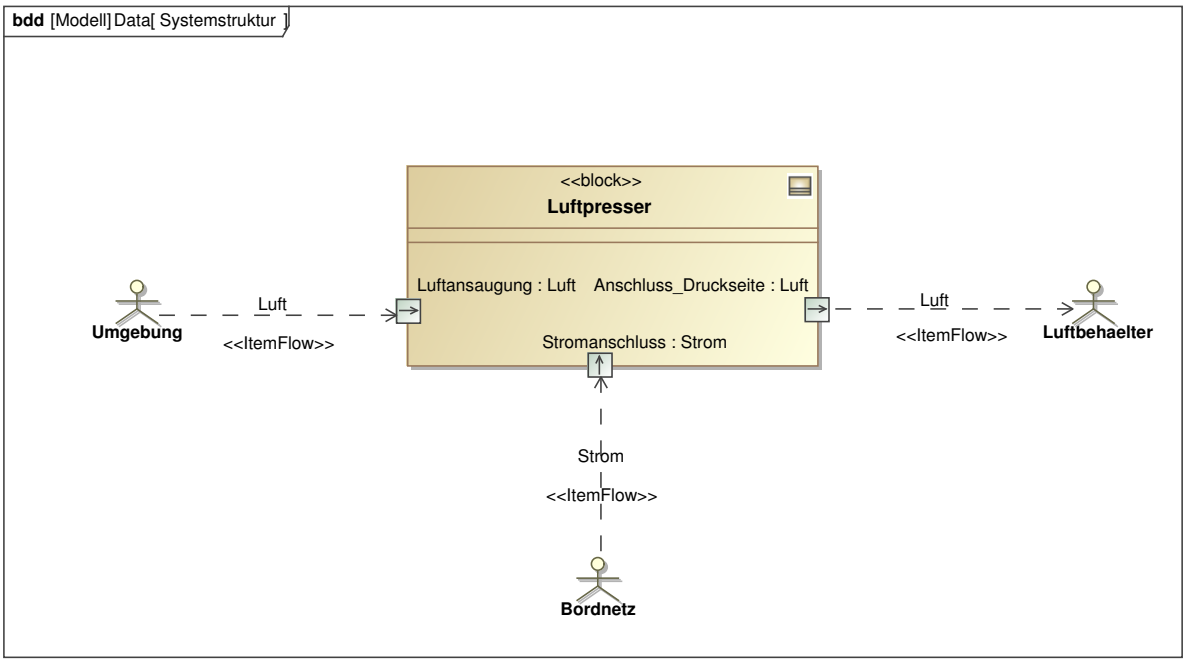


Abbildung 5.14.: Implizite Modellierung von Objektflüssen (Struktur)

Eine ähnliche Modellierung ist auch möglich, wenn anstelle der Aktivitätsparameter Objektknoten verwendet werden. Diese Alternative bietet sich an, wenn eine Aktivität keine Parameter besitzt. Auch hier kann durch eine Allokationsbeziehung des Objektknotens auf ein entsprechendes Strukturartefakt (Objektflussport, Block) die Ausgabe eines Objektstroms dargestellt werden.

Beispiele für die Anwendung dieses Modellierungsmusters im Bahnbereich sind:

- Kontinuierliche Übertragung von Informationen über das LZB-Kabel zwischen Streckenzentrale und Fahrzeug
- Signal eines Dopplerradars zur Geschwindigkeitsmessung
- Ölflüsse in einem Federspeicherbremssystem
- Modellierung von Bauteilen der Druckluftbremsanlage (siehe obiges Beispiel)

Das Senden von Objekten muss sich konsistent in das übrige Anforderungsmodell einfügen. Es gelten daher sinngemäß die gleichen Bedingungen wie beim Versenden von Signalen.

Empfang von Signalen

Das Gegenstück zum Signalversand ist der Empfang von Signalen, für dessen Modellierung sich die *AcceptEventAction [OMG07-2, S. 309 ff.]* eignet. Diese Aktion erwartet einen oder mehrere Events und setzt die Ausführung entlang der wegführenden Kontroll- oder Objektflüsse erst dann fort, wenn das zugeordnete Ereignis eingegangen ist. Durch dieses Konstrukt kann somit die weitere Aktivitätsausführung vom Eingang eines bestimmten Signals abhängig gemacht werden. Eine *AcceptSignalAction* lässt sich über eine Allocate-Beziehung einem Port oder einem Block zuordnen, wodurch ausgedrückt wird, dass das entsprechende Ereignis am bezeichneten Port oder von der referenzierten Komponente als Quelle erwartet wird. Analog zu der SendSignalAction repräsentiert dieses Konstrukt den zeitdiskreten Empfang von Signalen.

Syntaktisch bildet die Schwalbenschwanz-ähnliche Form der *AcceptSignalAction* ein Gegenstück zur pfeilförmigen *SendSignalAction*, was der Lesbarkeit der Diagramme entgegen kommt. Das folgende Bildbeispiel zeigt die Verwendung der *AcceptSignalAction* im Kontext einer Bahnübergangssicherungsanlage. Diese erwartet den Eingang des Signals „Sicherungsanstoss" und setzt dann erst den Ablauf entlang der ausgehenden Kontrollflusskante fort. Damit wird modelliert, dass die Aktivität „BÜ sichern" erst dann ausgeführt wird, wenn das Signal zum Sicherungsanstoß eingegangen ist.

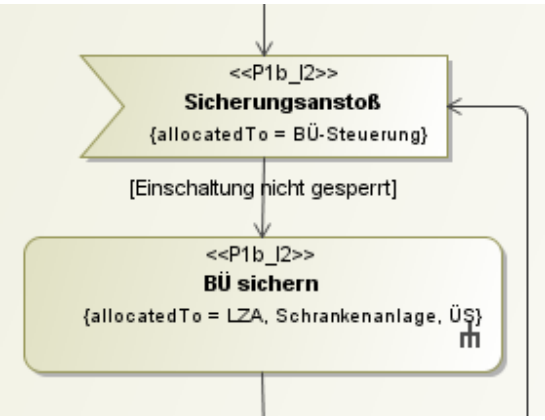


Abbildung 5.15.: Beispiel für Signalempfang

Auch für das Empfangen von Signalen gelten Konsistenzbedingungen gegenüber den übrigen Modellelementen. Diese sind ähnlich den Bedingungen beim Versenden von Signalen, nur dass die Kommunikationsrichtung umgekehrt ist (von der Umgebung zum System).

Timer

Dem Empfang von Signalen sehr ähnlich ist das Abwarten einer bestimmten Zeitbedingung

innerhalb einer Aktivität. Realisiert wird die Zeitabhängigkeit über die *AcceptTimeEventAction [OMG07-2, S. 238 ff.]* (Zeitereignis-Aktion), die semantisch gesehen eine Spezialisierung der *AcceptEventAction* durch spezielle *at*()- und *after*()-Trigger ist. Die Art des Triggers legt dabei fest, ob eine bestimmte Zeitspanne (*after()*-Trigger) oder ein bestimmter Zeitpunkt (*at()*-Trigger) abgewartet werden soll. Erreicht ein Token die Zeitereignis-Aktion, wird dessen Weitergabe an die ausgehende Kante verzögert, bis der trigger aktiv wird. Syntaktisch existiert für die Zeitereignisse eine eigene Notation in Form einer Sanduhr.

In Bild 5.16 wird die Verwendung einer Zeitereignis-Aktion gezeigt, bei der durch einen *after()*-Trigger der weitere Ablauf des Kontrollflusses um 12 Sekunden verzögert wird.

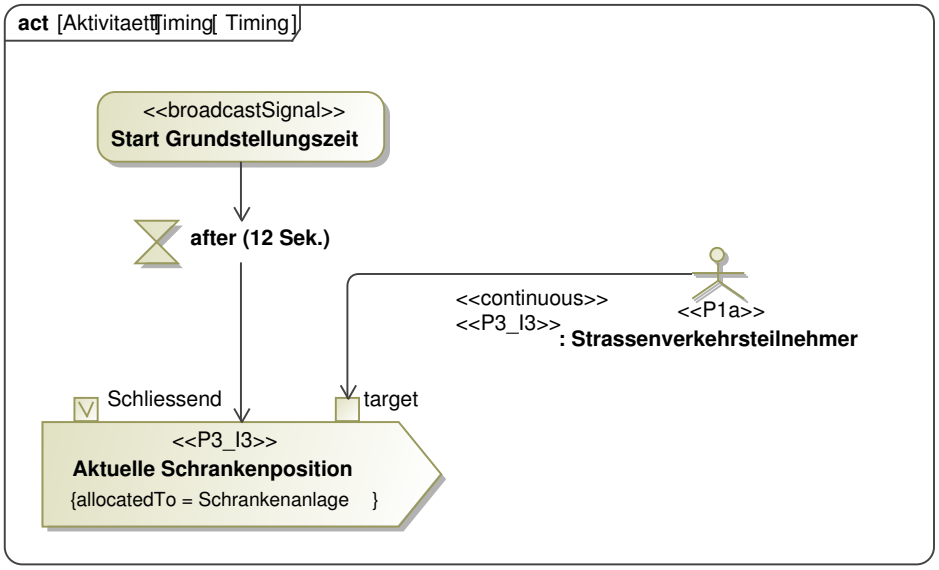


Abbildung 5.16.: Verwendung einer Zeitereignis-Aktion

Dieses Modellierungskonstrukt kann für Anforderungsmodelle immer dann eingesetzt werden, wenn konkrete, zeitliche Vorgaben für Abläufe gemacht werden sollen, beispielsweise

- bei der Modellierung der zeitlichen Randbedingungen der PZB (Dauer zwischen Beeinflussung und Bedienung der Wachsamkeitstaste, Dauer der Unterschreitung der Umschaltgeschwindigkeit)
- in Verbindung mit schleifenförmigen Kontrollflüssen kann eine periodische Abarbeitung von bestimmten Aktionen modelliert werden, z.B. die in bestimmten Zeitabständen wiederkehrende Übertragung von Zugpositionen an das RBC im ETCS-Level 2

Entscheidungskonstrukte

Weiteres wichtiges Element bei der Verhaltens-Modellierung sind Entscheidungskonstrukte, also Verzweigungen im Kontrollfluss, bei denen anhand der Auswertung eines booleschen Ausdrucks über den weiteren Ablauf entschieden wird. Aktivitätsdiagramme bieten zwei verschiedene Möglichkeiten an, Verzweigungen zu modellieren:

- Über *guards* an entsprechenden Kontrollflüssen und *decision nodes* (Entscheidungsknoten)
- Über die Verwendung von *conditional nodes* (bedingten Knoten)

Die erste Methode erzeugt kompaktere und einfacher lesbare Diagramme, wenn wenige Verzweigungen vorhanden sind, erkauft diesen Vorteil aber mit der Notwendigkeit von Ausdrücken

in einer Aktionssprache an den Kontrollflusskanten (siehe 5.1.4.4.4). Das zweite Vorgehen kann zwar vollständig auf vordefinierte UML-Aktionen zurückgreifen, die Notation ist allerdings weniger intuitiv zu begreifen und belegt - zumindest bei einer kleinen Zahl an Verzweigungs-Alternativen - erheblich größere Flächen im Diagramm. Beide Varianten werden im Folgenden mit ihren jeweiligen Anwendungsgebieten diskutiert.

Guards sind Ausdrücke, die sich in eine boolesche Aussage auflösen lassen und einer Kante des Kontrollflusses zugeordnet sind. Ist der Ausdruck wahr, kann ein Token die Kante passieren, ist der Ausdruck falsch, ist die Kante für Token gesperrt. Zusammen mit Entscheidungsknoten lassen sich somit Strukturen aufbauen, die den weiteren Ablauf hinter dem Entscheidungsknoten vorgeben. Dies zeigt Bild 5.17.

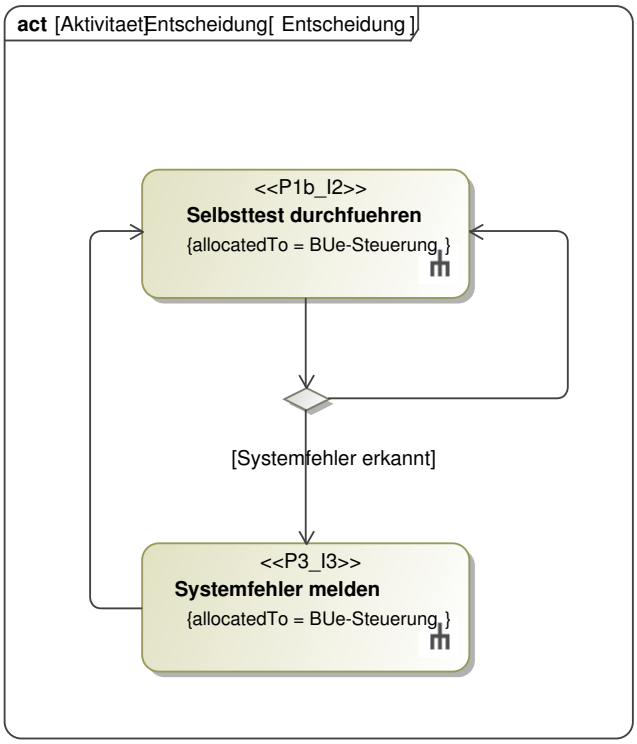


Abbildung 5.17.: Entscheidungsstrukturen mit guards und Entscheidungsknoten

Im dort dargestellten Beispiel wird im Kontrollfluss zu verschiedenen Folgeaktivitäten verzweigt, je nach dem, ob ein Systemfehler aufgetreten ist, oder nicht. Anzumerken ist allerdings, dass die Ausdrücke der *guards* hier in natürlicher Sprache formuliert sind. Für ein ausführbares Modell müsste jedoch eine Sprache wie z.B. die OCL verwendet werden.

Diese Art der Modellierung ist insbesondere dann geeignet, wenn sich der Kontrollfluss nur in wenige Zweige[3] aufspaltet und entweder keine Ausführbarkeit erforderlich ist oder aber sich für die *guards* einfache Ausdrücke in einer Aktionssprache ableiten lassen. Sind hingegen komplexe Aktivitätsabläufe zur Ermittlung der booleschen Aussage erforderlich, eignet sich eher die im Folgenden beschriebene Vorgehensweise mit bedingten Knoten.

Ein bedingter Knoten besteht dabei aus mehreren *clauses* (Klauseln), die jeweils aus einer *test section* und einer *body section* bestehen. In der *test section* muss genau eine Aktivität einen booleschen Wert an einem speziellen *result*-Pin bereitstellen. Ist der am Pin anstehende Wert wahr,

[3] Zwei, maximal drei ausgehende Zweige scheinen eine sinnvolle Grenze darzustellen

wird die zugehörige *body section* ausgeführt. Das nachfolgende Bild 5.18 zeigt eine komplexe Entscheidungsstruktur in Form eines bedingten Knoten.

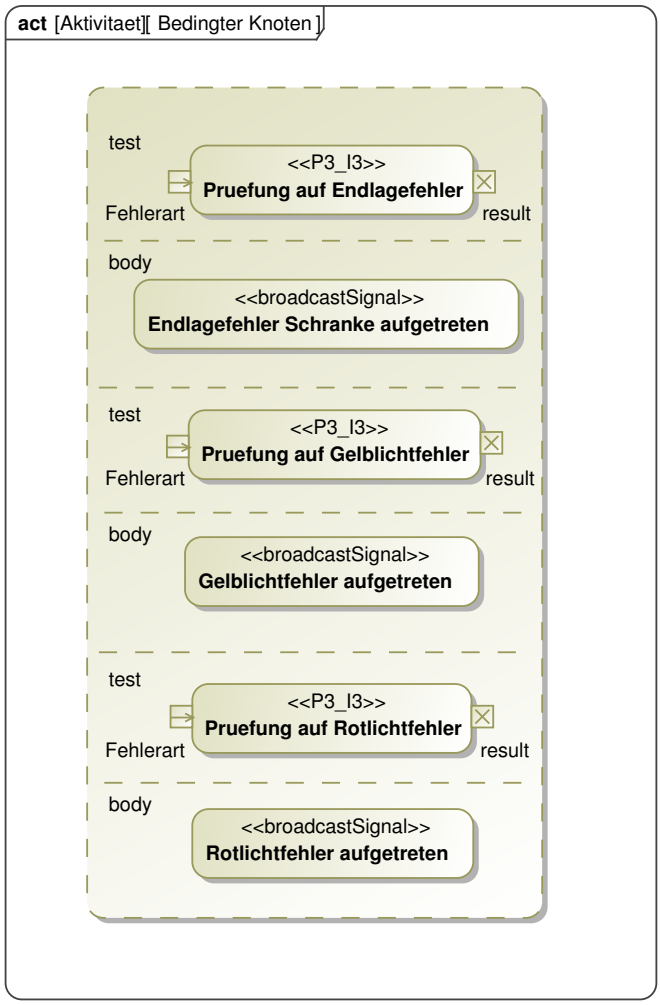


Abbildung 5.18.: Entscheidungsstrukturen mit bedingten Konten

Im Beispiel sind drei Klauseln enthalten, bei der jeweils in der *test section* auf eine bestimmte Fehlerart geprüft wird. Ist das Ergebnis am *result*-Pin wahr, war also die abgeprüfte Fehlerart aufgetreten, wird die entsprechende Fehlermeldung per *BraodcastSignalAction* versendet. Anschließend wird die Ausführung der Aktivität hinter dem bedingten Knoten fortgesetzt. Vorteil dieser Modellierung ist der Verzicht auf Ausdrücke in Aktionssprache und die übersichtlichere Darstellung auch vieler verschiedener Alternativen. Einen gravierenden Unterschied gibt es jedoch für den weiteren Aktivitätsablauf: Da die Ausführung immer an der gleichen, wegführenden Kante fortgesetzt wird, eignet sich dieses Konstrukt nur dann, wenn zwar abhängig von einem Testergebnis verschiedene Aktivitäten ausgeführt werden sollen, aber anschließend für alle Alternativen der gleiche weitere Ablauf vorgesehen ist. Es lassen sich also nicht alle Entscheidungs- und Verzweigungskonstrukte mit bedingten Knoten modellieren.

5.1.4.4.6. Zuteilung des Verhaltens auf Ports und Subsysteme

Ebenfalls Bestandteil der Systemverhaltenssicht ist die Zuteilung des Verhaltens auf Elemente

der Subsystemarchitektur (siehe 5.1.4.5.2). Eine Zuteilung weist dabei einem Verhaltenselement (beispielsweise einer Aktion) ein Strukturelement (beispielsweise einen Port oder eine Subsystemkomponente) zu und drückt darüber aus, welcher Teil des Systems für die Realisierung des Verhaltens verantwortlich ist. Im Modell erfolgt die Zuteilung durch eine *allocate*-Beziehung vom Verhaltens-Element auf das Struktur-Artefakt. Die allocate-Beziehung kann laut SysML-Spezifikation eben genau für den beschriebenen Zweck verwendet werden:

> „Behavior allocation relates to the systems engineering concept segregating form from function. This concept requires independent models of "function" (behavior) and "form" (structure), and a separate, deliberate mapping between elements in each of these models." [OMG07-1, S. 133]

Einige der in 5.1.4.4.5 beschriebenen Verhaltenskonstrukte benötigen diese Zuteilung zwingend, für andere Elemente ist sie optional. Zwingend erforderlich ist eine Zuteilung bei der Modellierung des kontinuierlichen Versendens von Informations- oder Stoffströmen. Erst durch die Verknüpfung eines Objektknotens mit einem Strukturartefakt kann ausgedrückt werden, dass ein kontinuierlicher Strom dieses Objektes über die entsprechende Komponente verschickt wird.

Weiterhin muss unterschieden werden zwischen Zuteilungen auf Portdefinitionen des SuB und der Zuteilung auf Subsysteme:

- Eine Zuteilung auf Portdefinitionen des SuB ist dann erforderlich, wenn im betrachteten Teilmodell keine weitere Unterteilung in Subsysteme vorgenommen wird, aber Verhaltenskonstrukte zum Versenden von Signalen oder Objekten vorhanden sind. Jedes betroffene Verhaltenskonstrukt muss dann auf denjenigen Port oder FlowPort zugewiesen werden, über den das Signal oder der Objektstrom versendet werden soll.
- Existiert im aktuellen Teilmodell eine Subsystemarchitektur, dann müssen alle Aktivitäten auf die entsprechenden Subsysteme alloziiert werden. Sollte dabei festgestellt werden, dass eine Aktivität mehr als einer Subsystemkomponente zugewiesen werden müsste, ist dies ein Indiz dafür, dass die Aktivität in weitere Subaktivitäten aufgeteilt werden sollte, bis eine eindeutige Zuteilung möglich ist. Dadurch, dass alle Portdefinitionen des SuB in der Subsystemarchitektur auf Subsysteme delegiert werden (siehe 5.1.4.5.2), ist mit der Alloziierung einer Aktivität auf eine entsprechende Subsystemkomponente auch implizit eine Alloziierung auf den per *delegate*-Beziehung verbundenen Port enthalten.

Eine sehr intuitiv zu verwendende Methode zur Zuordnung von Verhaltensartefakten auf Strukturelemente besteht in der Verwendung von *AllocateActivityPartitions,* auch Swimlanes genannt. Dabei handelt es sich um rechteckige Bereiche innerhalb eines Verhaltensdiagramms, die jeweils ein Strukturelement repräsentieren. Befindet sich ein Verhaltenselement innerhalb eines dieser Bereiche, wird automatisch eine allocate-Beziehung zur zugehörigen Komponente erzeugt. In der SysML-Spezifikation wird die Funktion der *AllocateAcitvityPartitions* folgendermaßen definiert:

> „AllocateActivityPartition is used to depict an «allocate» relationship on an Activity diagram. The AllocateActivityPartition is a standard UML2::ActivityPartition, with modified constraints [...]." [OMG07-1, S. 134]

Im Anforderungsmodell ist die Verwendung von Swimlanes zwar empfohlen, aber optional. Eine Allocate-Beziehung kann auch manuell über die entsprechenden Funktionen des verwendeten Modellierungswerkzeugs erstellt werden. Bild 5.19 zeigt die Nutzung von *AllocateAcitvityPartitions* zur Zuteilung von Aktivitäten.

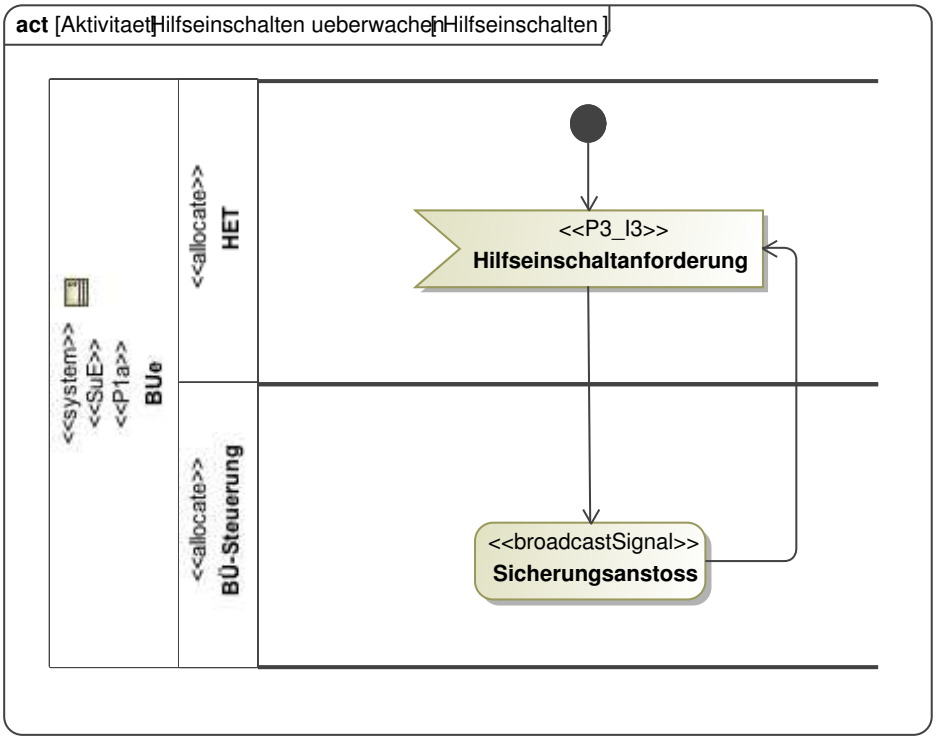


Abbildung 5.19.: Zuteilung von Aktivitäten zu Strukturelementen per AllocateActivityPartition

5.1.4.4.7. Zusammenfassung Tabelle 5.5 fasst die vorgestellten Modellierungsmuster, deren Anwendungsgebiete und Eigenschaften zusammen. Der entsprechende Ausschnitt aus dem Metamodell ist in Bild 5.20 dargestellt.

Verhaltensmuster	SysML-Sprachelemente	Anmerkungen
Senden von Signalen	SendSignalAction	Geeignet für zeitdiskretes Versenden von Signalen. Die Aktion wird durch Input-Pins parametrisiert, um den Empfänger der Nachricht und etwaige Attribute festzulegen. Die Menge der versendeten und empfangenen Signale muss der Menge der Signale in der Systemabgrenzungssicht, der Systemfunktionssicht, und der Szenariensicht enthalten sein (siehe 5.1.5.1) Wenn keine Subsystemarchitektur vorliegt, muss dieses Element einem Port oder FlowPort des Systems zugeordnet werden, ansonsten einem Teil der Subsystemarchitektur (siehe 5.1.5.5).
Senden von Objekten	SendObjectAction Objektknoten und Allocations auf Strukturartefakte	Die SendObjectAction ist geeignet für das zeitdiskrete Versenden von Objekten. Kontinuierliche Ströme von Objekten können über Aktivitätsparameter bzw. Objektknoten und eine Zuordnung auf Strukturartefakte modelliert werden. Es gelten sinngemäß die gleichen Konsistenzbedingungen wie beim Nachrichtenversand.
Empfangen von Signalen	AcceptEventAction	Die AcceptEventAction hält den Kontrollfluss bis zum Eintreffen eines bestimmten Signals an. Dadurch kann die Reaktivität eines Systems auf Signaleingang modelliert werden. Die Menge der versendeten und empfangenen Signale muss der Menge der Signale in der Systemabgrenzungssicht, der Systemfunktionssicht, und der Szenariensicht enthalten sein (siehe 5.1.5.1) Wenn keine Subsystemarchitektur vorliegt, muss dieses Element einem Port oder FlowPort des Systems zugeordnet werden, ansonsten einem Teil der Subsystemarchitektur (siehe 5.1.5.5).
Timer	AcceptTimeEventAction	Die AccpetTimeEventAction hält den Kontrollfluss bis zum Erreichen eines absoluten Zeitpunktes oder bis zum Ablauf einer Zeitspanne an. Es kann somit das Zeitverhalten eines Systems abgebildet werden.
Entscheidungskonstrukte	Decision Node und Guards oder Conditional Nodes	Beide Modellierungsmuster erlauben die Modellierung von Verzweigungen im Aktivitätsablauf. Bei einfachen Verzweigungen oder Verzicht auf Ausführbarkeit sind Decision Nodes mit Guards das geeignetere Beschreibungsmittel. Conditional Nodes kommen ohne Ausdrücke in Aktionssprache aus, erlauben aber nur Entscheidungspfade, die wieder auf einen ausgehenden Zweig konvergieren.
Zuteilung von Verhaltenskonstrukten auf Strukturartefakte	AllocateActivityPartition	Jede AllocateAcitvityPartition repräsentiert ein Strukturartefakt. Ist eine Subsystemarchitektur vorhanden, müssen alle Aktivitäten einem Subsystem zugeordnet werden. Andernfalls müssen SendSignalActions, AcceptSignalEvents und Objektknoten einem Port des SuB zugeordnet werden.

Tabelle 5.5.: Modellierungselemente für die Verhaltensmodellierung

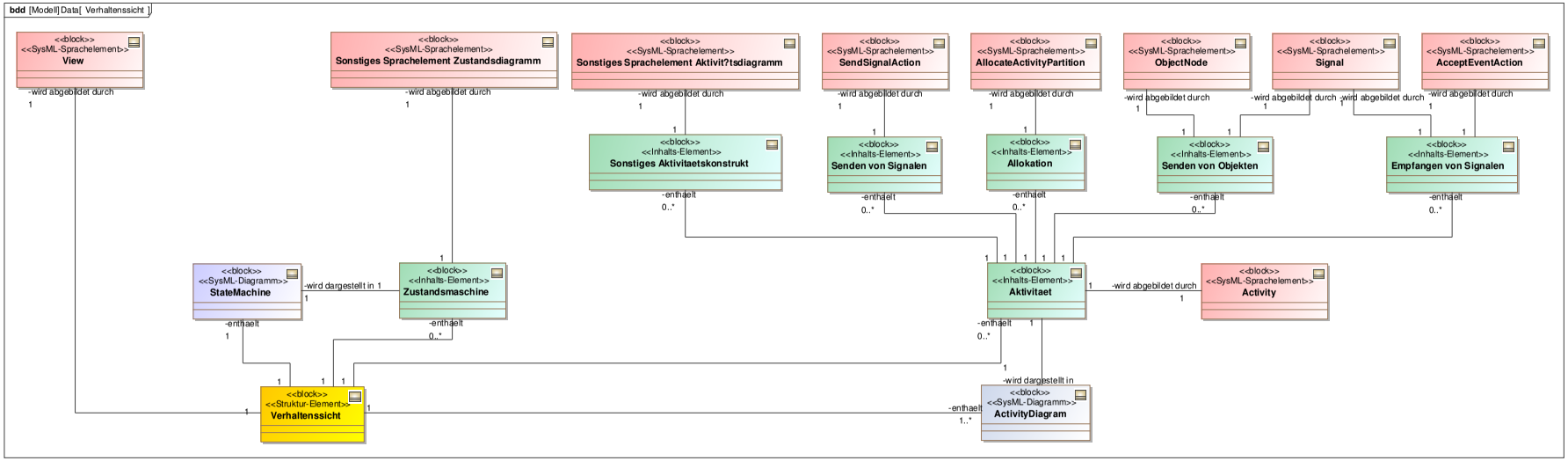


Abbildung 5.20.: Metamodell-Definition der Verhaltenssicht

5.1.4.5. Subsystem-Struktur/Whitebox-Sicht

Gemeinsam war den in den vorherigen Kapiteln vorgestellten Sichten eine Blackbox-Perspektive auf das SuB, das als monolithischer Block ohne Angaben zu dessen innerer Struktur betrachtet wurde. In diesem Abschnitt wird nun die Sicht auf die Interna der Blackbox beschrieben. Diese interne Sicht auf die einzelnen Komponenten des SuB dient gleichzeitig als Schnittstelle zur Modellierung der nächst tieferen Detaillierungsebene entsprechend dem in Abschnitt 5.1.3.1 beschriebenen Gliederungskonzept. Die Subsystemsicht ist daher zur Umsetzung des Ebenenkonzepts zwingend nötig.

5.1.4.5.1. Begriffsdefinitionen

Folgt man den Grundsätzen einer rein funktionalen Spezifikation, ist eine ausschließliche Blackbox-Darstellung der Systemanforderungen ausreichend, da die Whitebox-Spezifikation eine Aufgabe des Herstellers ist. Prinzipiell ist dieser Gedanke nachvollziehbar, weil so die Menge der möglichen Lösungen für das in der Anforderungsspezifikation beschriebene Problem nicht unnötig eingeschränkt wird. Auch hat jeder Hersteller damit die Freiheit, eine an seine Produkt- und Entwicklungsstrategie angepasste Systemarchitektur zu entwickeln - der nicht spezifizierte Leerraum kann mit eigenen Ideen und Konzepten ausgefüllt werden. Wenn dieses Paradigma auch bei der Software-Entwicklung durchaus sinnvoll ist, ist es bei der Anforderungsspezifikation von komplexen Systemen aus nachfolgend aufgeführten Gründen nicht immer zielführend.

Bei der Entwicklung von Software ist aus Auftraggeber-Sicht die Funktionalität die wesentliche Eigenschaft. Die Bedeutung der Architektur der Software tritt hinter die funktionalen Eigenschaften zurück, da an die einzelnen, internen Software-Komponenten meist keine Kundenanforderungen gestellt werden. Zudem benötigt Software im Gegensatz zu Hardware per se keine physische Struktur, verschleißt nicht und bedarf keiner regelmäßigen Wartung zur Erhaltung der Verfügbarkeit[4]. Anders ist dies bei heterogenen Systemen. Insbesondere wenn mechanische, verschleißbehaftete Bauteile wie Relais, Schalter, Motoren, Leuchtmittel oder ähnliches eingesetzt werden, kann es sinnvoll sein, auch die Systemarchitektur in der Anforderungsspezifikation zu beschreiben. Hat sich beispielsweise eine bestimmte Aufteilung der logischen Systemfunktionen auf eine physische Systemstruktur bewährt, kann es sinnvoll für den Kunden sein, diese bei Neuanschaffungen in der Systemanforderungsspezifikation vorzugeben[5]. Ähnliches gilt auch für die Bevorratung von Ersatzteilen und die gesamte Instandhaltungslogistik. Hier liegt das Einsparpotenzial in der Verwendung von bestimmten Komponenten, die schon in großer Stückzahl an anderer Stelle beim Betreiber verwendet werden. Das Anforderungsmodell muss somit auch eine Möglichkeit zur Spezifikation der internen Architektur des SuB vorsehen. Diese Whitebox-Sicht auf das SuB muss dabei die folgenden Anforderungen erfüllen:

- Definition von *Subsystemkomponenten* innerhalb des SuB
- Darstellung der *Subsystemarchitektur* innerhalb des SuB

Die Subsysteme bilden weiterhin die Menge an Artefakten, auf die das in der Blackbox-Sicht modellierte Verhalten zugeteilt werden kann. Dieser Vorgang wird in Abschnitt 5.1.4.4.6 beschrieben.

[4] Davon ausgenommen sind Änderungen an der Software zur Beseitigung von Fehlern und von Sicherheitslücken

[5] Für eine Grubenbahn in stark staubiger Umgebung kann es beispielsweise sinnvoll sein, möglichst viele Komponenten der Leit- und Sicherungstechnik an einer zentralen, abgeschotteten Stelle zu konzentrieren, um übermäßiger Verschmutzung vorzubeugen. In anderen Fällen mag hingegen eine möglichst verteilte Systemarchitektur sinnvoll sein.

5.1.4.5.2. Modellierung der Subsystemsicht mit der SysML

Die Subsystemsicht besteht aus zwei wesentlichen Elementen: Der Definition der Subsystemkomponenten und der Zusammenstellung dieser Komponenten zu einer Subsystemarchitektur. Die Definition der Subsystemkomponenten erfolgt dabei auf Blockebene, was bedeutet, dass diese zunächst als generische Komponenten ohne Betrachtung ihrer spezifischen Verwendung in einem System modelliert werden. Die konkrete Rolle, die eine Subsystemkomponente im betrachteten Teilsystemkontext übernimmt, wird dann in der Systemarchitektursicht festgelegt. Verfügt eine Bahnübergangssicherungsanlage beispielsweise über zwei Radsensoren, die als Einschalt- bzw. Ausschaltkontakt verwendet werden, so ist die abzubildende Subsystemkomponente der Radsensor. Einschalt- und Ausschaltkontakt sind die konkreten Verwendungen dieser Komponente in einem speziellen Systemkontext[6].

Geeignetes SysML-Sprachelement für die Subsystemkomponenten sind Blöcke, die mit dem Stereotyp *Subsystem* gekennzeichnet werden. Um zu zeigen, dass sich die Subsystemkomponenten in der Gliederungshierarchie unterhalb des jeweils zugehörigen SuB befinden, sollten sie als dessen innere Elemente modelliert werden.

Die Zusammenstellung der so definierten Subsystemkomponenten zur Subsystemarchitektur erfolgt über *internal block diagrams* (interne Blockdiagramme). Diese stellen die SysML-Variante der aus der UML bekannten *composite structure diagrams* (Kompositionsstrukturdiagramme) dar und dienen einem ähnlichen Zweck, der in der UML-Spezifikation folgendermaßen definiert ist:

> „A composite structure diagram depicts the internal structure of a classifier, as well as the use of a collaboration in a collaboration use." [OMG07-2, S. 191]

Insbesondere der erste Teil dieser Spezifikation entspricht der im Folgenden beschriebenen Nutzung der internen Blockdiagramme. Über die Forderungen der UML hinausgehend, erzwingt die SysML-Spezifikation eine strengere Zuordnung eines internen Blockdiagramms zum übergeordneten Block. So heisst es in [OMG07-1, S. 40]:

> „The diagram heading name for an internal block diagram [...] must identify the name of a SysML block as its modelElementName. [...] All the properties and connectors that appear inside the internal block diagram belong to the block that is named in the diagram heading name."

Im Anforderungsmodell wird diese Vorgabe durch die Modellierung des internen Blockdiagramms als inneres Element des SuB umgesetzt, wodurch alle im internen Blockdiagramm hinzugefügten Strukturen automatisch auch Teil des übergeordneten SuB sind. Das SuB selbst wird im internen Blockdiagramm durch den Diagrammrahmen repräsentiert. Alle Ports des SuB werden durch Portsymbole auf der Rahmenlinie dargestellt. Als Konsistenzbedingung (formale Definition in 5.1.5) muss gelten, dass alle Ports, die in der Systemabgrenzungssicht modelliert wurden, auch in der Subsystemsicht enthalten sind. In Bild 5.21 sind beispielsweise die Ports „p_Schrankenanlage" und „p_LZA" Beispiele für die Ports des SuB, die in der Subsystemsicht dargestellt werden.

Die einzelnen Subsysteme werden durch *parts* (Teile) abgebildet, die eine spezifische Nutzung einer Subsystemkomponente darstellen. Teile sind deren Instanzen und entsprechen somit den Objekten im Kompositionsstruktur-Diagramm der UML. Konkret erfolgt die Instanziierung durch die Typ-Zuweisung einer der definierten Subsystemkomponenten zum jeweiligen Teil. Im obigen Beispiel des Bahnüberganges existieren im internen Blockdiagramm beispielsweise die Teile „Einschaltkontakt" und „Ausschaltkontakt", beide vom Typ „Radsensor". Teile können über Multiplizitäten verfügen, die angeben, wie oft ein bestimmtes Teil im konkreten Subsystementwurf

[6] Der Radsensor könnte in einem anderen Kontext beispielsweise auch als Achszählpunkt verwendet werden

enthalten ist. So verfügt die beispielhafte Bahnübergangssicherungsanlage - je nach Konfiguration - über 4 bis 6 Lichtzeichenanlagen für die Straßenverkehrsteilnehmer. Im Modell wird dies über ein Teil „Lichtzeichenanlage" vom Typ „LzA" mit der Multiplizitätsangabe „[4...6]" dargestellt. Bild 5.21 zeigt die oben beschriebenen Konstrukte.

Zur Subsystemarchitektur gehört weiterhin eine Darstellung der Verbindungen der einzelnen Subsystemkomponenten untereinander und mit den Ports des SuB. Dazu bietet sich der aus der UML übernommene *connector* (Konnektor) an, der eine zunächst unspezifische Verbindung zweier Komponenten repräsentiert. In [OMG07-2, S. 175] heißt es:

> „[Connector] specifies a link that enables communication between two or more instances. [...] The link may be realized by something as simple as a pointer or by something as complex as a network connection. In contrast to associations, which specify links between any instance of the associated classifiers, connectors specify links between instances playing the connected parts only."

Diese allgemeine Definition eines Verbindungselements ist sehr gut geeignet für die Verwendung in einem Anforderungsmodell, da so implizite Annahmen über die Realisierung der Verbindung ausgeschlossen werden können. Differenziert wird allerdings die Funktion des Konnektors über ein Typ-Attribut, für das es zwei Ausprägungen gibt: Den *delegate*-Konnektor und den *assembly*-Konnektor. Die erste Funktion wird in der UML-Spezifikation definiert als:

> „A delegation connector is a connector that links the external contract of a component (as specified by its ports) to the internal realization of that behavior by the component's parts. It represents the forwarding of signals (operation requests and events): a signal that arrives at a port that has a delegation connector to a part or to another port will be passed on to that target for handling." [OMG07-2, S. 154]

Ein Konnektor muss demnach dann vom Typ delegate sein, wenn er Stoff- oder Informationsflüsse von einem Port an der Systemgrenze zur verantwortlichen Subsystemkomponente weiterleitet. Dies ist im Bild 5.21 auf der nächsten Seite beispielsweise beim Konnektor vom Port „p_Einschaltsensor" zur Komponente „ES" der Fall. Diese Verknüpfung beider Elemente sagt aus, dass die Komponente „ES" diejenige Funktionalität innerhalb des SuB realisiert, die jenes über den Port „p_Einschaltsensor" seiner Umwelt anbietet.

Die assembly-Ausprägung eines Konnektors hingegen dient der Subsystem-internen Verbindung zweier funktional abhängiger Teile, wobei der eine Teil dem anderen Teil eine benötigte Dienstleistung anbietet. In der UML-Spezifikation heißt es:

> „An assembly connector is a connector between two components that defines that one component provides the services that another component requires." [OMG07-2, S. 154]

Ein solches Verhältnis besteht beispielsweise zwischen BÜ-Steuerung und Einschaltsensor. Die BÜ-Steuerung benötigt für das Einschalten der Sicherungsanlage die Information über einen sich näherndes Schienenfahrzeug. Der Teil „ES" stellt diese Funktionalität bereit und liefert entsprechende Ausgaben an die BÜ-Steuerung. Daher sind beide Teile über ein assembly-Konnektor miteinander verbunden.

Mit den beschriebenen Modellelementen sind die ersten beiden in 5.1.4.5.1beschriebenen Anforderungen an eine Subsystemsicht erfüllt. Die Zuteilung des Systemverhaltens auf die einzelnen Subsystemkomponenten hingegen erfolgt nicht in den Diagrammen der Subsystemsicht, sondern in den Verhaltensdiagrammen der Systemebene durch Swimlanes. Diese Konstrukte sind in 5.1.4.4.6 beschrieben.

Die verwendeten Modellelemente sind in Tabelle 5.6 zusammengefasst.

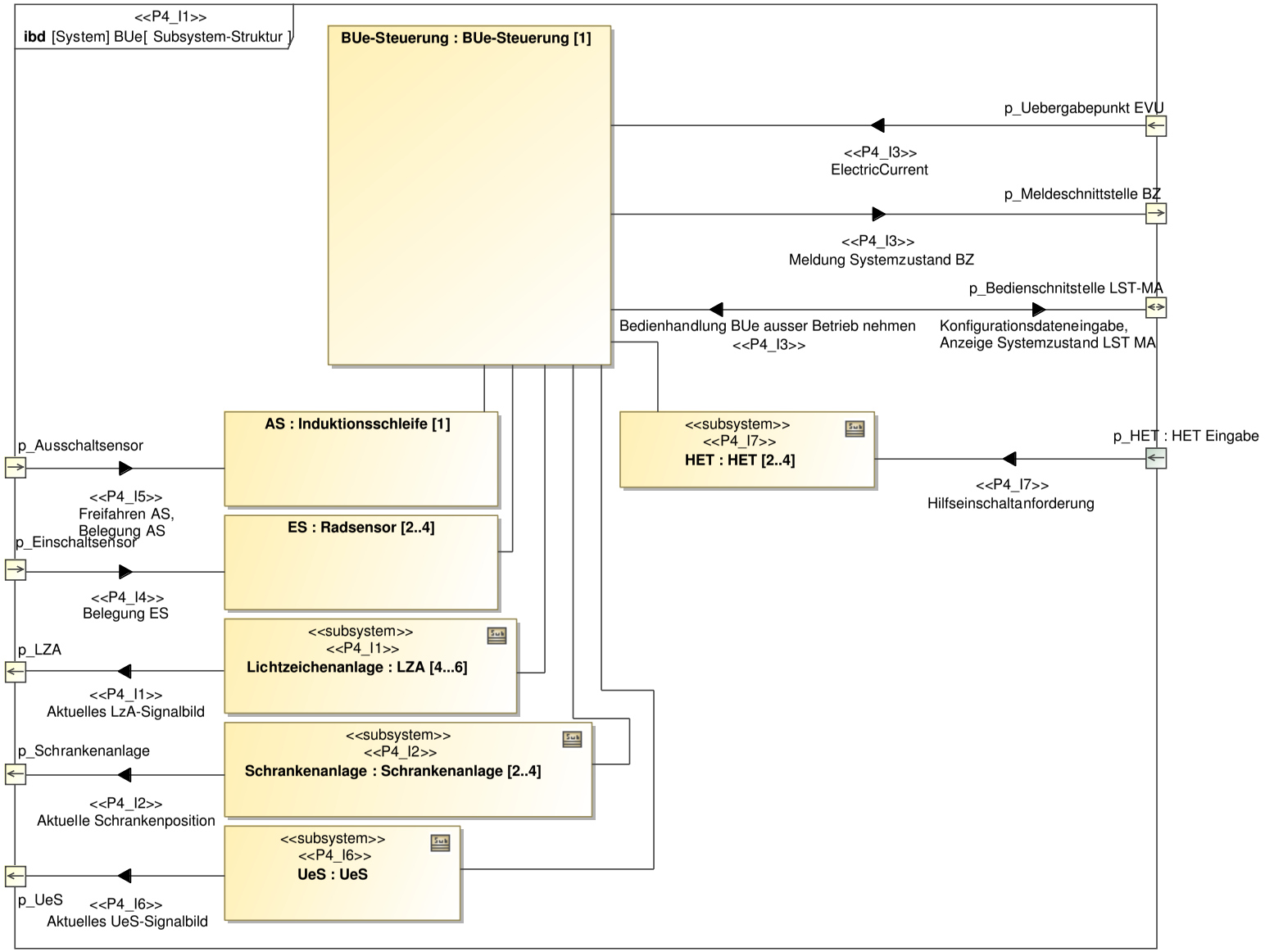


Abbildung 5.21.: Beispielhafte Subsystemstruktur

Inhaltselement	**SysML-Sprachelemente**	**Anmerkungen**
Diagrammart	*internal block diagram*	*Diagramm muss als inneres Element des SuB modelliert werden*
SuB	Diagrammrahmen des internal block diagram	
Ports des SuB	Ports bzw. FlowPorts am Diagrammrahmen	Konsistenzkriterium: Die Menge der Ports muss der Menge der Ports in der Subsystemsicht entsprechen (siehe 5.1.5.4).
Generische Definition der Subsystemkomponenten	Block mit Stereotyp Subsystem	Beschreibt eine Subsystemkomponente auf Klassenebene unabhängig von deren konkreten Verwendung innerhalb eines SuB.
Konkrete Nutzungen der Subsystemkomponenten	Parts	Instanziierung einer allgemeinen Subsystemkomponente für eine konkrete Verwendung innerhalb der Subsystemarchitektur. Durch Multiplizitäten kann die Anzahl an gleichen parts vorgegeben werden.
Verbindungen zwischen Ports des umgebenden SuB und Subsystemkomponenten	Connector, Typ delegate	Verbindet die Ports an der Systemgrenze mit den verantwortlichen parts. Der verbundene Part muss die Funktionalität besitzen, die Informationsflüsse über den Port zu erzeugen bzw. zu verarbeiten. Die über den Connector ausgetauschten Informationen werden über Signale beschrieben Konsistenzkriterium: Die Menge der Signale muss der Menge der Signale in der Systemabgrenzungssicht, der Systemfunktionssicht, der Szenariensicht und der Systemverhaltenssicht entsprechen (siehe 5.1.5.1).
Verbindungen zwischen Subsystemkomponenten untereinander	Connektor, Typ assembly	Verbindung der einzelnen parts, wobei eine funktionale Abhängigkeit zwischen den parts besteht.

Tabelle 5.6.: Modellierungselemente für die Subsystemarchitektur

5.1.4.6. Anforderungsschnittstellensicht

Die bisher beschriebenen Sichten stellten in unterschiedlichen Perspektiven das funktionale Modell dar, wodurch die funktionalen Systemanforderungen definiert werden. Wie bereits in Abschnitt 4 dargelegt und im nachfolgenden Abschnitt 5.2.1 detailliert ausgeführt, umfasst die gesamte Anforderungsspezifikation jedoch auch text-basierte, nicht-funktionale Anforderungen und Schnittstellen zu externen, text-basierten Dokumenten. Diese besitzen Berührungspunkte mit dem funktionalen Modell. In der in diesem Unterabschnitt beschriebenen Anforderungsschnittstellensicht werden diese Berührungspunkte berücksichtigt und in einer SysML(A)-konformen Notation dargestellt.

Nach dem in 5.2.3 beschriebenen Prinzip sollen die text-basierten Informationen außerhalb des funktionalen Modells verwaltet werden, wobei Verknüpfungspunkte zwischen beiden Teilen

der Anforderungsspezifikation vorgesehen sind. Die Anforderungsschnittstellensicht repräsentiert die Verknüpfungspunkte in Form von einzelnen SysML-*Requirements* [OMG07-1, S. 146] in einem Requirement-Diagram, wobei diese innerhalb des funktionalen Modells nicht verändert werden können. Sie dienen gewissermaßen als „Brückenköpfe" für die Verknüpfung der textbasierten Informationen mit dem funktionalen Modell. Durch die Verwendung verschiedener Arten von Beziehungen können verschiedene semantische Aussagen transportiert werden. So kann beispielsweise ausgedrückt werden:

- welche nicht-funktionalen Anforderungen welchen Elementen des funktionalen Modells zugeordnet sind
- welche Anforderungen aus externen Dokumenten im funktionalen Modell berücksichtigt worden sind (Darstellung der Nachverfolgbarkeit externer Anforderungen)
- welche Szenarien der Szenariensicht als Testfälle für externe Anforderungen verwendet werden können

Die folgende Tabelle zeigt die in der SysML(A) für die Anforderungsschnittstellensicht definierten Beziehungen, ihre Verwendung und die zugehörige semantische Aussage.

Beziehung	**Beteiligte Elemente**	**Erklärung**
DeriveReqt [OMG07-1, S. 149 f.]	Zwischen Requirements	Zeigt an, dass ein Requirement von einem anderen Requirement abgeleitet wurde, z.B. ein detailliertes Requirement von einer übergeordneten Vorschrift.
Containment	Zwischen Requirements	Zeigt an das ein Unter-Requirement in einem übergeordneten Requirement enthalten ist.
Satisfy [OMG07-1, S. 151 f.]	Zwischen Requirement und Element des funktionalen Modells (nicht bei Szenarien)	Zeigt an, dass ein bestimmtes Modellelement ein Requirement erfüllt.
Trace [OMG07-2, S. 691]	Zwischen beliebigen Elementen	Zeigt an, dass ein Element den gleichen Sachverhalt ausdrückt wie ein Element an anderer Stelle des gleichen oder eines anderen Modells. Dadurch lässt sich beispielsweise zeigen, dass sich eine externe Anforderung in einem internen Requirement wiederfindet. Dieser Stereotyp drückt keine Erfüllung von Requirements aus.
Verify [OMG07-1, S. 152 f.]	Zwischen Requirement und Szenario	Zeigt an, dass ein Szenario als Testfall für eine Anforderung dienen kann
Dependency	Zwischen Requirement und Element des funktionalen Modells	Zeigt an, dass eine nicht-funktionale Anforderung einem bestimmten Element des funktionalen Modells zugeordnet ist, z.B. ein Umgebungstemperaturbereich einer Komponente des Systems

Tabelle 5.7.: Mögliche Beziehungen zwischen Requirements und Modellelementen

Die typische Nutzung einiger der dargestellten Beziehungen zeigt Abbildung 5.22.

Zu erkennen ist, wie sich die Inhalte einer Passage einer Konzernrichtlinie bis zu den entsprechenden funktionalen Modellelementen nachverfolgen lassen. Für die zusichernde Behörde (wie

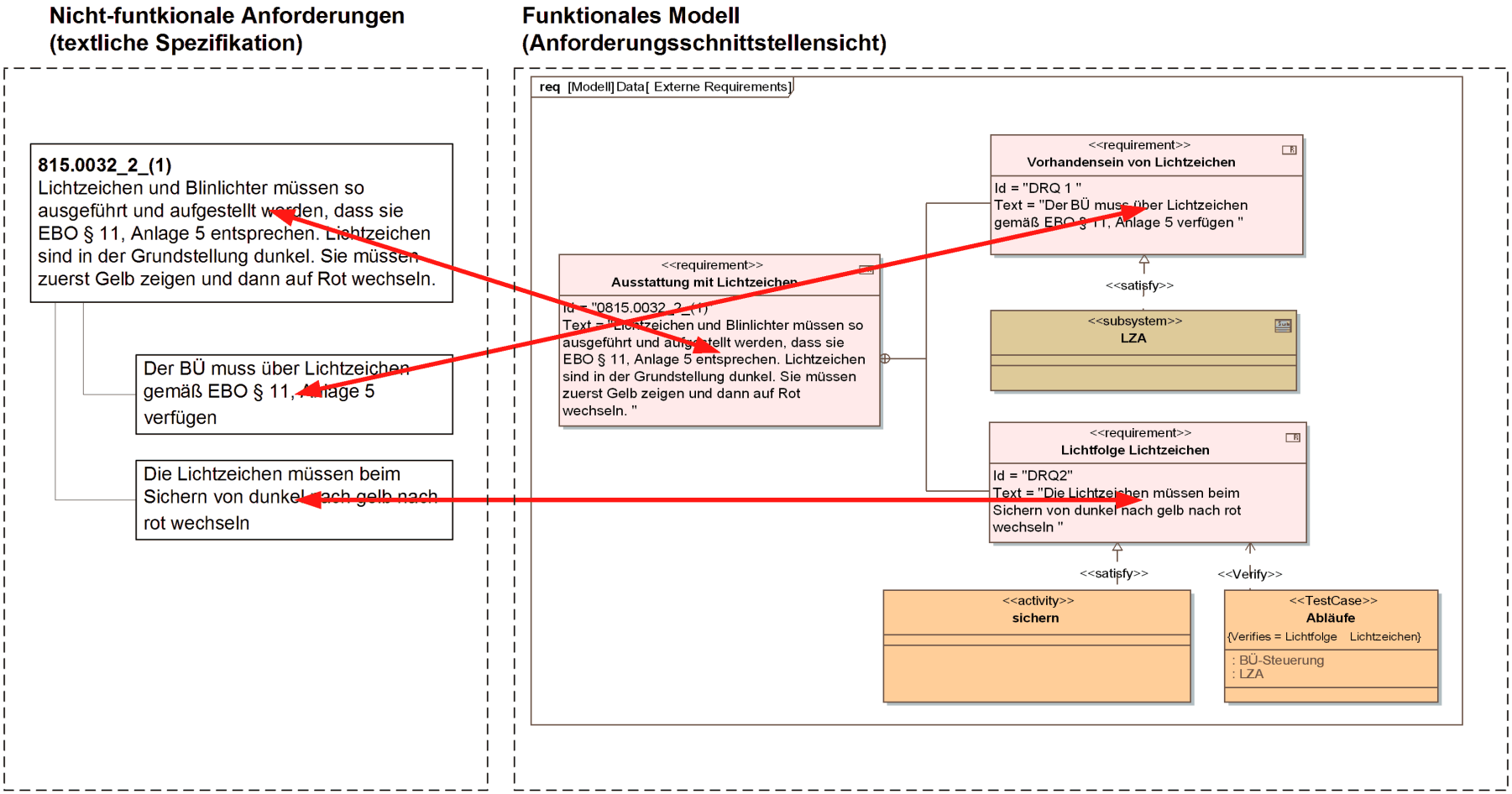


Abbildung 5.22.: Beispielhafte Einbindung externer, natürlich-sprachlicher Dokumente

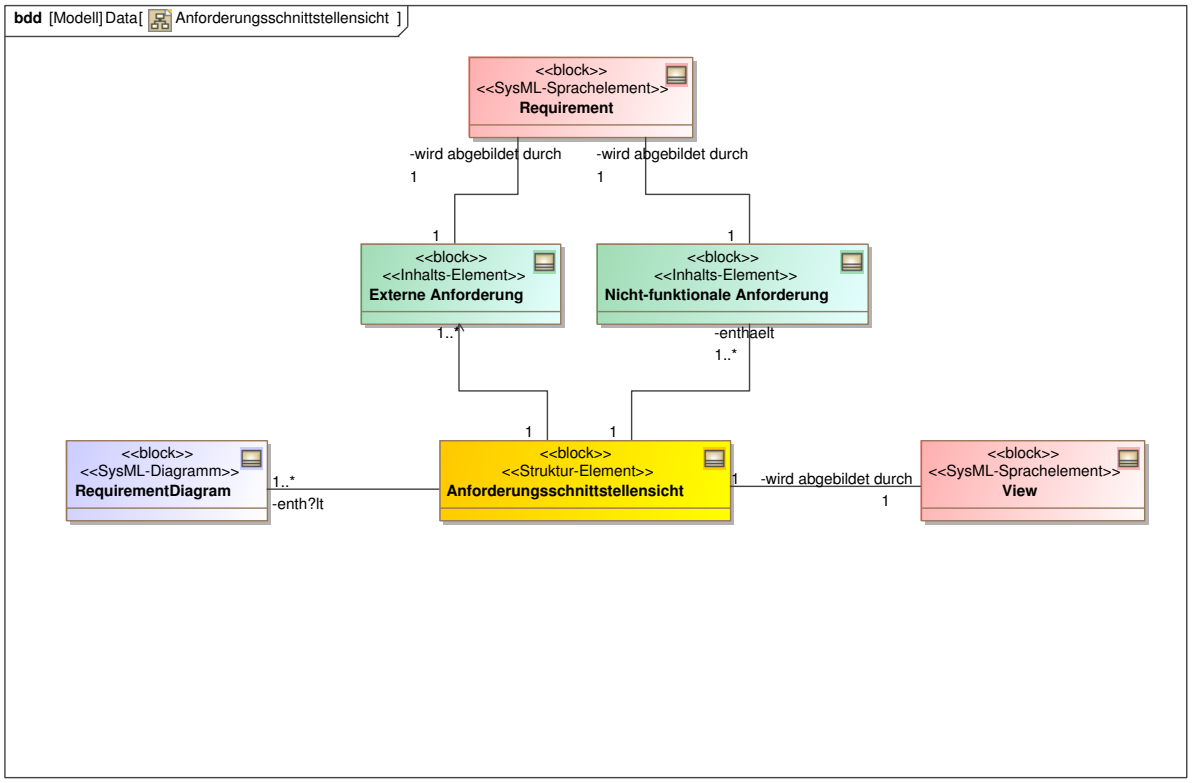


Abbildung 5.23.: Metamodell-Definition der Anforderungsschnittstellensicht

z.B. das EBA) kann somit gezeigt werden, dass die in den relevanten Richtlinien enthaltenen Informationen im Anforderungsmodell korrekt abgebildet wurden. Konkret handelt es sich dabei um Satz (1), Absatz 2 des Moduls 0032 der DB-Konzernrichtlinie 815 [KOR815], das die Ausstattung eines BÜ mit Lichtzeichenanlagen regelt und deren prinzipielles Verhalten definiert. Analysiert man den Textabschnitt, so zeigt sich, dass dieser eigentlich zwei atomare Anforderungen enthält: Einerseits, dass der BÜ über eine EBO-konforme Lichtzeichenanlage verfügen muss, andererseits, dass diese bei der BÜ-Sicherung eine bestimmte Signalfolge einhalten muss. Daher werden aus der Ursprungspassage in der Richtlinie zunächst die beiden enthaltenen Aussagen extrahiert und mit dem originalen Textfragment verknüpft. Dies geschieht außerhalb des funktionalen Modells, beispielsweise in IBM/telelogic DOORS während der Durchführung des entsprechenden Subprozesses S8, der in Abschnitt 6.4.8 beschrieben wird.

Bei der Synchronisierung der textlichen Anforderungen mit dem funktionalen Modell werden die einzelnen Textelemente in SysML-Requirements umgesetzt, wobei die ermittelten Unter-Anforderungen über eine *Containment*-Beziehung verknüpft werden. Anschließend werden - innerhalb des funktionalen Modells - diejenigen Elemente mit den Requirements über eine *<<satisfy>>*-Beziehung verbunden, die das entsprechende Requirement erfüllen. Szenarien, die als Testfälle dienen, werden mit dem zugehörigen Requirement über eine *<<verify>>*-Beziehung verknüpft. Zur besseren Verständlichkeit erfolgt diese Verknüpfung im oben dargestellten Beispiel direkt innerhalb des Requirement-Diagramms.

Die Spezifikation der Anforderungsschnittstellensicht im Metamodell kann Abbildung 5.23 entnommen werden.

5.1.5. Konsistenzregeln

Für die in den Kapiteln 5.1.4.1 bis 5.1.4.5 beschriebenen verschiedenen Sichten auf das Anforderungsmodell lassen sich Konsistenzregeln aufstellen, deren Einhaltung ein Auseinanderdriften des Informationsgehalts in den einzelnen Sichten verhindern soll. Diese Regeln gelten zunächst unabhängig von den zur Modellierung verwendeten Werkzeugen, wobei durch die Funktionen der üblichen UML-Modellierungssoftware eine Einhaltung der Regeln oftmals automatisch erzwungen wird. So verwendet diese üblicherweise eine Datenbank zur Speicherung der im Modell verwendeten Elemente. Die in den einzelnen Diagrammen dargestellten Symbole sind dann meist nur Referenzen auf die Objekte in der Datenbank. Somit ist sichergestellt, dass ein Element mit dem gleichen Bezeichner in allen Diagrammen den gleichen Informationsgehalt aufweist, wodurch ein Teil der Konsistenzregeln erfüllt wird. Andere Regeln können jedoch nur durch entsprechende Sorgfalt bei der Modellierung und durch ständiges Prüfen des Modells auf Regelkonformität berücksichtigt werden. Aus diesem Grund sind in der Prozessdefinition entsprechende Kontroll-Aktivitäten vorgesehen. Einige Modellierungswerkzeuge bieten jedoch durch Skriptsprachen oder ähnliche Konstrukte die Möglichkeit, die Einhaltung der Regeln weitgehend zu automatisieren.

Im folgenden werden die modellweit geltenden Konsistenzregeln durch semi-formale Ausdrücke in UML und durch formale mathematische Ausdrücke spezifiziert.

5.1.5.1. Konsistenz der Signale

In allen Sichten auf das funktionale Modell muss die gleiche Menge an Signalen enthalten sein, da in allen Sichten Signale zur Darstellung der ausgetauschten Informationen verwendet werden, wie der Ausschnitt des Metamodells in Bild 5.24 zeigt. Mathematisch lässt sich dieser Sachverhalt folgendermaßen ausdrücken: Wenn $S = \{S_1, S_2, \ldots, S_n\}$ die Menge aller im Modell enthaltenen Signale ist, dann muss für die Mengen der Signale

- $S_{Systemfunktionssicht}$ in den Diagrammen der Systemfunktionssicht (als übertragene Informationen über die Informationsflüsse)
- $S_{Systemabgrenzungssicht}$ in den Diagrammen der Systemabgrenzungssicht (als übertragene Informationen über die Informationsflüsse)
- $S_{Szenariensicht}$ in den Diagrammen der Szenariensicht (als übertragene Informationen der Nachrichten)
- $S_{Verhaltenssicht}$ in den Diagrammen der Verhaltenssicht (als Parameter der *SendSignalActions* und *AcceptEventActions*)
- $S_{Subsystemsicht}$ in den Diagrammen der Subsystemsicht (also übertragene Informationen über die delegate-Konnektoren

gelten:

$$S_{Systemfunktionssicht} = S_{Systemabgrenzungssicht} = S_{Szenariensicht} = S_{Verhaltenssicht} = S_{Subsystemsicht} = S \tag{5.1}$$

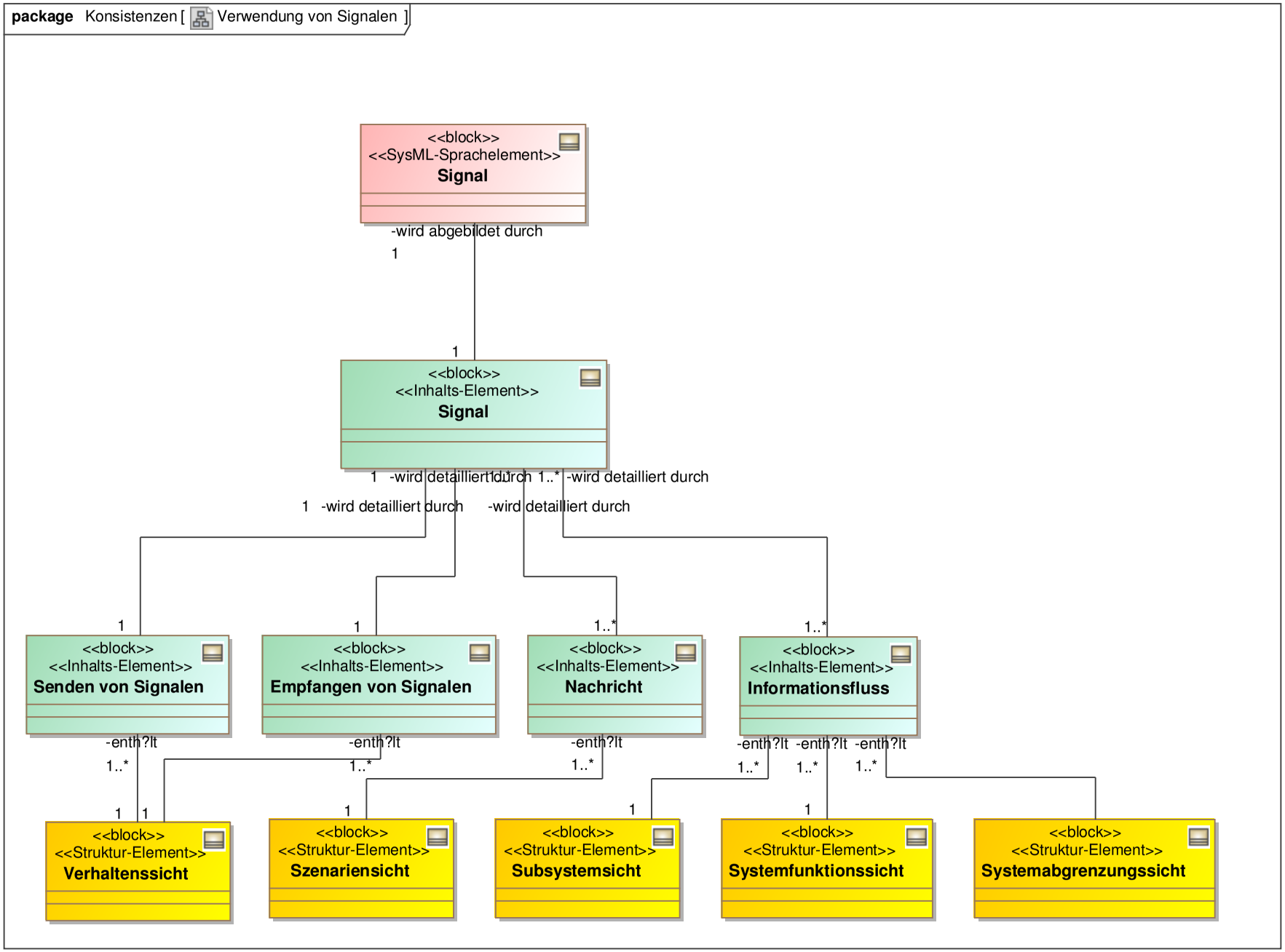


Abbildung 5.24.: Verwendung von Signalen in allen Sichten

5.1.5.2. Konsistenz der Akteure

Die im Modell abgebildete Menge an Akteuren muss in allen Sichten konsistent verwendet werden. Ist $A = \{A_1, A_2, \ldots, A_n\}$ die Menge der Akteure im Anforderungsmodell muss für die Mengen der in den einzelnen Sichten benutzten Akteure

- $A_{Systemfunktionssicht}$ in den Diagrammen der Systemfunktionssicht (als Artefakte für externe Akteure und Systeme)
- $A_{Systemabgrenzungssicht}$ in den Diagrammen der Systemabgrenzungssicht (als Artefakte für externe Akteure und Systeme)
- $A_{Szenariensicht}$ in den Diagrammen der Szenariensicht (als Lebenslinien für externe Akteure und Systeme)

gelten

$$A_{Systemfunktionssicht} = A_{Systemabgrenzungssicht} = A_{Szenariensicht} = A \tag{5.2}$$

Für die Akteure in den Verhaltensdiagrammen (als Ziele der *SendSignalActions*) $A_{Verhalten}$ gilt abweichend

$$A_{Verhalten} \subseteq A \tag{5.3}$$

da nur für die ausgehenden Signale Akteure als Parameter benötigt werden. Die Menge der verwendeten Akteure in den Verhaltensdiagrammen ist daher immer kleiner, maximal gleich der Menge an Akteuren in den anderen Sichten. Für Akteure muss weiterhin gelten, dass jeder Akteur mindestens zu einem Anwendungsfall in der Systemfunktionssicht (5.1.4.2) eine Assoziation aufweisen muss. Wäre dies nicht der Fall, hätte der Akteur keine Beziehung zu den Systemfunktionen und könnte entfallen. Dieser Sachverhalt wird im Struktur-Metamodell durch eine Abhängigkeit zwischen Akteuren, Anwendungsfällen und den sie verbindenden Assoziationen dargestellt (Bild 5.25). Jeder Akteur ist demnach mindestens über eine Assoziation mit mindestens einem Anwendungsfall verknüpft.

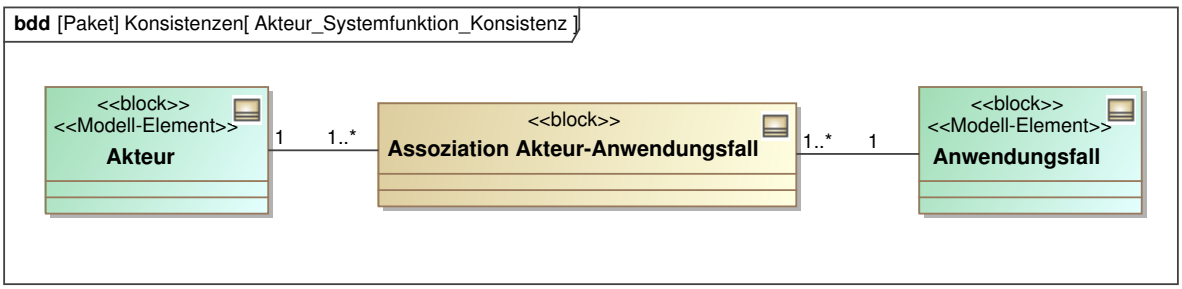


Abbildung 5.25.: Abhängigkeit zwischen Akteuren und Anwendungsfällen

5.1.5.3. Konsistenz der Informationsflüsse

Die modellierten Informationsflüsse zwischen System und Umgebung werden in der Systemabgrenzungssicht und in der Systemfunktionssicht verwendet. In beiden Sichten muss die gleiche

Menge an Informationsflüssen verwendet werden. Daher muss für die Mengen $I_{Systemabgrenzungsicht}$ und $I_{Systemfunktionssicht}$ gelten

$$I_{Systemabgrenzungssicht} = I_{Systemfunktionssicht} = I \tag{5.4}$$

wenn $I = \{I_1, I_2, \ldots, I_n\}$ die Menge aller Informationsflüsse im Modell ist.

5.1.5.4. Konsistenz der Ports

In der Systemabgrenzungssicht und in der Subsystemsicht muss die gleiche Menge an Ports verwendet werden. Ist die Menge der Ports im Anforderungsmodell definiert als $P = \{P_1, P_2, \ldots, P_n\}$, so muss für die Menge an Ports

- $P_{Systemabgrenzungssicht}$ in den Diagrammen der Systemabgrenzungssicht
- $P_{Subsystemsicht}$ in den Diagrammen der Subsystemsicht

gelten:

$$P_{Systemabgrenzungssicht} = P_{Subsystemsicht} = P \tag{5.5}$$

5.1.5.5. Konsistenz der Allokationen

Bei der Zuteilung von Aktivitäten auf Strukturartefakte wird gemäß 5.1.4.4.6 zwischen zwei Fällen unterschieden: Existiert keine Subsystemarchitektur, werden Aktionen zum Senden von Objekten, Objektknoten und Aktionen zum Empfang von Signalen den zugehörigen Port-Definitionen zugeteilt. Die Metamodell-Abbildung dieses Konstruktes zeigt Bild 5.26.

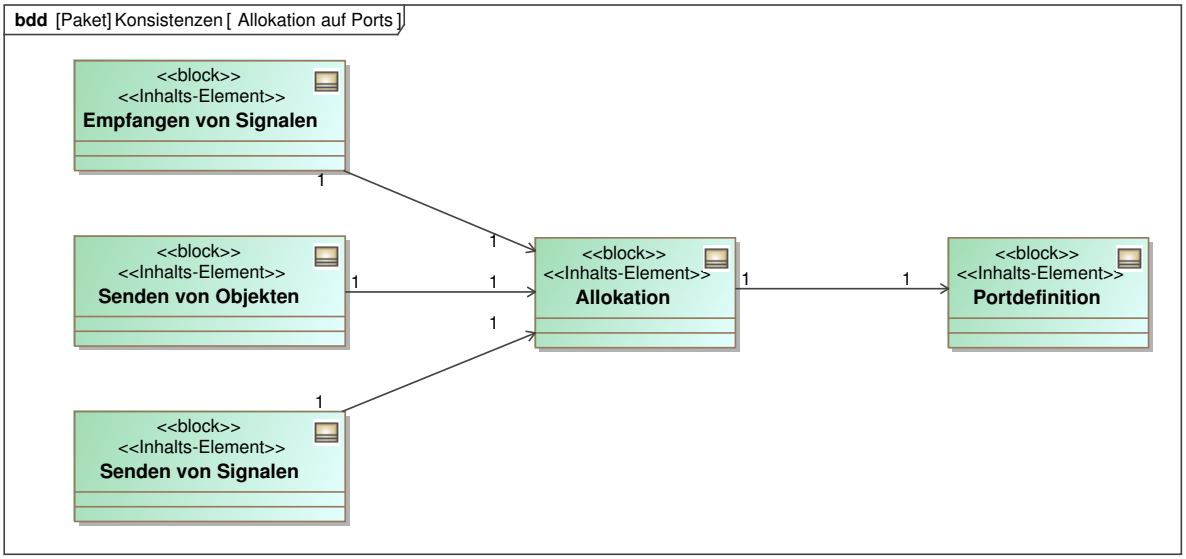


Abbildung 5.26.: Allokation auf Port-Definitionen

Im anderen Fall werden alle Aktivitäten den entsprechenden Teilen der Subsystemarchitektur zugewiesen, die semi-formale Spezifikation dieses Konstruktes kann Bild 5.27 entnommen werden.

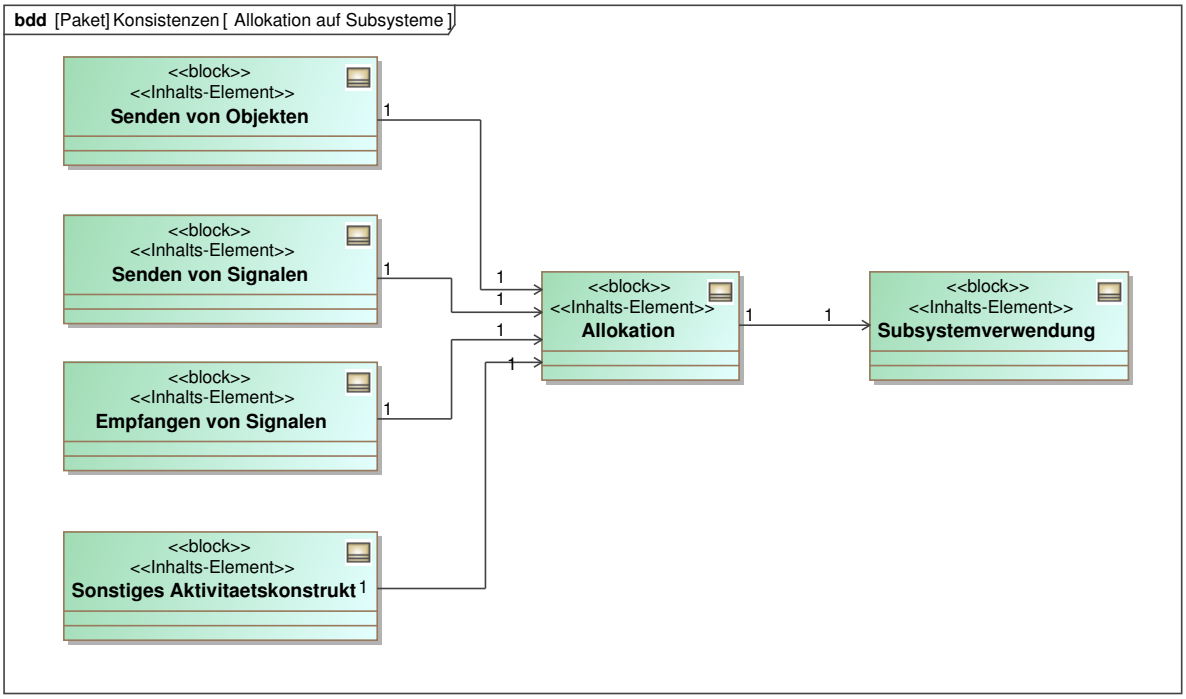


Abbildung 5.27.: Allokation auf Subsystem-Artefakte

5.2. Nichtfunktionale Anforderungen und externe Dokumente

Nachdem im vorherigen Abschnitt das funktionale Modell detailliert vorgestellt wurde, wird im Folgenden - als zweite wesentliche Komponente der Anforderungsspezifikation - die Abbildung der nicht-funktionalen Anforderungen und die Verknüpfung mit Inhalten externer Dokumente beschrieben. Bezogen auf das Gesamtkonzept betrifft dies die in Abbildung 5.28 grau hervorgehobenen Bereiche.

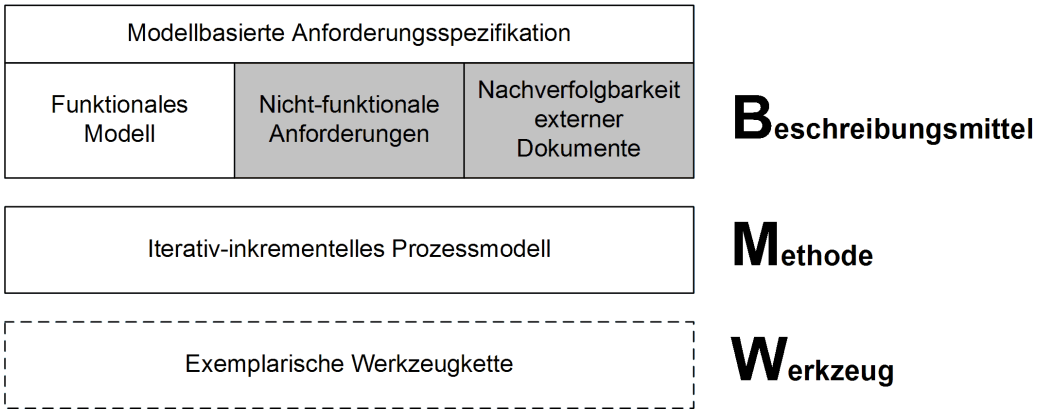


Abbildung 5.28.: Einordnung der nicht-funktionalen Anforderungen in den BMW-Kontext

5.2.1. Nicht funktionale Anforderungen

Neben den funktionalen Anforderungen, die im funktionalen Modell abgebildet werden, umfasst eine Anforderungsspezifikation üblicherweise auch zahlreiche nicht-funktionale Anforderungen. Diese legen fest, wie und unter welchen Bedingungen die spezifizierten Funktionen ausgeführt werden sollen. Je nach System, Einsatzumgebung und Anwendungsfall kann die Art der abzubildenden nicht-funktionalen Anforderungen deutlich unterschiedlich ausfallen. Dennoch lassen sich bestimmte Gruppen nicht-funktionaler Anforderungen für bahntechnische Systeme aus mehreren Quellen und Überlegungen ableiten:

- Die CENELEC-Norm 50126 [EN50126] legt mit dem RAMS-Konzept für „Reliability, Availability, Maintainability und Saftey" (Zuverlässigkeit, Verfügbarkeit, Wartbarkeit und Sicherheit) wesentliche Kategorien für nicht-funktionale Systemanforderungen fest. Dies sind beispielsweise tolerierte Gefährdungsraten (tolerable hazard rates, THR) oder die mittlere geforderte Betriebsdauer zwischen Ausfällen (mean time between failure, MTBF).
- Die DIN EN 50121 [EN50121] beschreibt Anforderungen an die elektromagnetische Verträglichkeit von Systemen im Bahnbereich. Auch aus dieser Norm lassen sich entsprechende nicht-funktionale Anforderungen an ein System ableiten, wie beispielsweise die maximalen Werte für elektromagnetische Abstrahlung.
- Gleiches gilt für die Spezifikation der Umweltbedingungen, in denen ein System operieren soll. Diese werden für bahntechnische Systeme in der DIN EN 50125 [EN50125] definiert und umfassen beispielsweise zulässige Temperaturbereiche und zu ertragende mechanische Belastungen.
- Weiterhin gehören auch projektspezifische Randbedingungen, wie beispielsweise Reaktionszeiten, Symboliken für Bedien- und Meldeeinrichtungen und Leistungsfähigkeitsanforderungen zu den nicht-funktionalen Anforderungen.

Diese Aufstellung zeigt, dass es sich dabei üblicherweise um Informationen handelt (mitunter nur um einzelne Zahlenwerte), die sich inhaltlich nur schwierig durch eine modellbasierte Sprache wie die SysML ausdrücken lassen. Vielmehr scheint dazu die Nutzung kurzer Textfragmente sinnvoll, zumal die in Abschnitt 3.1 aufgeführten Mängel der natürlichen Sprache wegen der geringen Komplexität kaum zu Tage treten. Allerdings muss es möglich sein, die nicht-funktionalen Anforderungen (beispielsweise Angaben zu THRs oder Umgebungsbedingungen) einzelnen Elementen des funktionalen Modells zuzuordnen, beispielsweise einem Subsystem oder einer Systemfunktion. Daher muss eine Methode gefunden werden, die nicht-funktionalen Anforderungen zwar inhaltlich textlich abzubilden, sie aber dennoch mit den Mitteln der SysML an eine Entität des funktionalen Modells anzukoppeln und in das funktionale Modell zu integrieren.

5.2.2. Externe Dokumente

Eine ähnliche Problemstellung wie für die nicht-funktionalen Anforderungen ergibt sich für die Nachverfolgbarkeit von Anforderungen, die in externen Dokumenten[7] enthalten sind. Im Eisenbahnwesen, aber auch in vielen anderen Fachdomänen, existieren für die Entwicklung sicherheitskritischer Systeme zahlreiche Randbedingungen in Form von Normen, Gesetzen und Verordnungen sowie unternehmensinternen Richtlinien. Ein durch das Anforderungsmodell beschriebenes System muss diesen Randbedingungen genügen, was gegenüber der zulassenden Stelle (beispielsweise dem Eisenbahnbundesamt) nachgewiesen werden muss. Die oben genannten Randbedingungen liegen dabei meist in Form natürlich-sprachlicher Dokumente vor und bilden eine Ebene externer, unbeeinflussbarer Anforderungen. Das Anforderungsmodell muss Mechanismen aufweisen, durch die die Erfüllung der in Textform beschriebenen Randbedingungen dargestellt werden kann. Weiterhin existieren oftmals schon vor Beginn der eigentlichen Modellierung informelle Sammlungen von Informationen, deren Nachverfolgbarkeit und korrekte Umsetzung ebenfalls nachgewiesen werden muss. Wie in Abschnitt 6.2.1 beschrieben wird, lassen sich auch diese Informationen wie die oben genannten externe Dokumente behandeln.

Bei beiden Arten von externen Dokumenten zeigen sich deutliche Analogien zu den in Abschnitt 5.2.1 beschriebenen nicht-funktionalen Anforderungen: Diese Informationsmengen können inhaltlich nicht sinnvoll durch die SysML abgebildet werden, weisen aber dennoch Beziehungen zum funktionalen Modell auf. Unterschiedlich ist nur die Herkunft der Informationen: Während die nicht-funktionalen Anforderungen und die Informationen aus dem Modellierungsvorlauf durch den Ersteller der Anforderungsspezifikation beeinflusst werden können, sind die Inhalte externer Dokumente meist unveränderbar. Prinzipiell ist die Berücksichtigung externer Anforderungen nur ein Spezialfall der nicht-funktionalen Anforderungen.

Im folgenden Abschnitt soll daher ein Konzept vorgestellt werden, wie nicht-funktionale Anforderungen und Fragmente externer Dokumente mit dem funktionalen Modell verknüpft werden können. Die gewählte Lösung soll sich dabei gut in das funktionale Modell integrieren. Diese Integration soll dabei so weitreichend sein, dass der Ersteller der Anforderungsspezifikation einem Auftragnehmer lediglich das funktionale Modell übergeben muss, um sowohl alle funktionalen, als auch alle nicht-funktionalen Anforderungen abzudecken. Dies bedeutet, dass die text-basierten Informationen ins funktionale Modell gespiegelt werden müssen, auch wenn sie außerhalb des Modells verwaltet werden.

Beschrieben wird im folgenden Kapitel ein grobes Konzept. Die tatsächliche Umsetzung und technische Ausgestaltung dieses Konzeptes kann dann Gegenstand weiterführender Arbeiten sein.

[7] Gemeint sind damit Dokumente, deren Inhalt nicht im Verantwortungsbreich des Erstellers der Anforderungsspezifikation liegt

5.2.3. Konzept zur Einbindung textbasierter Informationen

Die SysML sieht spezielle Konstrukte vor, um textliche Anforderungen darzustellen: Das *requirement*-Artefakt dient zur Einbindung einzelner textlicher Spezifikationen und Anforderungen, verschiedene Arten von Beziehungen ermöglichen Relationen zwischen Anforderungen und Modellelementen. Die Verwendung dieser Konstrukte setzt allerdings voraus, dass alle textlichen Anforderungen direkt im jeweiligen SysML-Modell verwaltet werden. Nachteilig ist dabei, dass die Requirements in der SysML durch stereotypisierte Klassen abgebildet [OMG07-1, S. 141] werden, innerhalb der die Anforderungen durch mehrere Freitextfelder natürlich-sprachlich definiert werden. Das Klassensymbol agiert dabei weitgehend nur als Hülle für den Text. Diese Darstellung wird jedoch bereits bei einer geringen Zahl an Requirements unübersichtlich, und häufige Änderungen sind nur unkomfortabel durchzuführen. Die Verwaltung der Anforderungen in einem dedizierten Requirements-Management-Werkzeug und damit der Verzicht auf die Integration der textlichen Anforderungen in das funktionale Modell hätte jedoch den Nachteil, dass eine direkte Verknüpfung von textlichen Requirements und Elementen des funktionalen Modells nicht mehr möglich ist und zudem vertragsrelevante Informationen durch verschiedene Datenquellen repräsentiert werden.

Anzustreben ist daher ein hybrides Konzept, bei dem die nicht-funktionalen Anforderungen zwar durch SysML-Requirements-Artefakte abgebildet werden, die Verwaltung des textlichen Inhalts jedoch außerhalb des Modells geschieht. Dies ermöglicht die Verwendung eines besser geeigneten Werkzeugs für diese Aufgabe, wie beispielsweise IBM/telelogic DOORS. So kann der Vorteil der Integration von funktionalen und nicht-funktionalen Anforderungen ausgenutzt werden, ohne die Nachteile einer geringeren Übersichtlichkeit in Kauf nehmen zu müssen. Dieses Vorgehen erfordert jedoch eine geeignete Synchronisierung zwischen den textlichen Anforderungen außerhalb und innerhalb des funktionalen Modells. Jede externe Änderung der Anforderungen muss dabei unmittelbar in die Menge der Requirements im funktionalen Modell übernommen werden.

Die beschriebene mögliche Umsetzung dieser Idee ist in Abbildung 5.29 dargestellt.

Zu erkennen sind in der Abbildung die folgenden Elemente:

- Auf der linken Seite ist ein Dokument mit einzelnen, natürlich-sprachlichen Anforderungen dargestellt, die durch unterschiedliche Farben hervorgehoben sind. Diese textlichen Informationen werden außerhalb des funktionalen Modells verwaltet. Dazu gehört auch, dass Beziehungen zwischen einzelnen Textfragmenten definiert und Passagen der Ursprungsdokumente (z.B. Konzernrichtlinien der DB AG) aufgegliedert werden können, wenn sie mehr als eine atomare Anforderung enthalten. Die grün, orange und lila dargestellten Passagen stehen dabei für Anforderungen in externen Dokumenten, die blau symbolisierten Textabschnitte für nicht-funktionale Anforderungen, die vom Modellierer selbst festgelegt wurden.
- Durch eine geeignete technische Umsetzung werden aus den textlichen Informationen automatisiert SysML-Requirements erstellt, die dann in das funktionale Modell eingebunden werden. Dabei werden die Verknüpfungen zwischen den einzelnen Textfragmenten in entsprechende SysML-Beziehungen umgesetzt. Diese Konstrukte sind in dem mittleren Requirements-Diagramm dargestellt.
- Zwischen den SysML-Requirements und den Elementen des funktionalen Modells können dann ebenfalls Beziehungen modelliert werden, beispielsweise um die Nachverfolgbarkeit externer Anforderungen oder die Zuordnung bestimmter nicht-funktionaler Anforderungen zu Modellbestandteilen darzustellen. Beispielhafte Verknüpfungen sind zwischen den

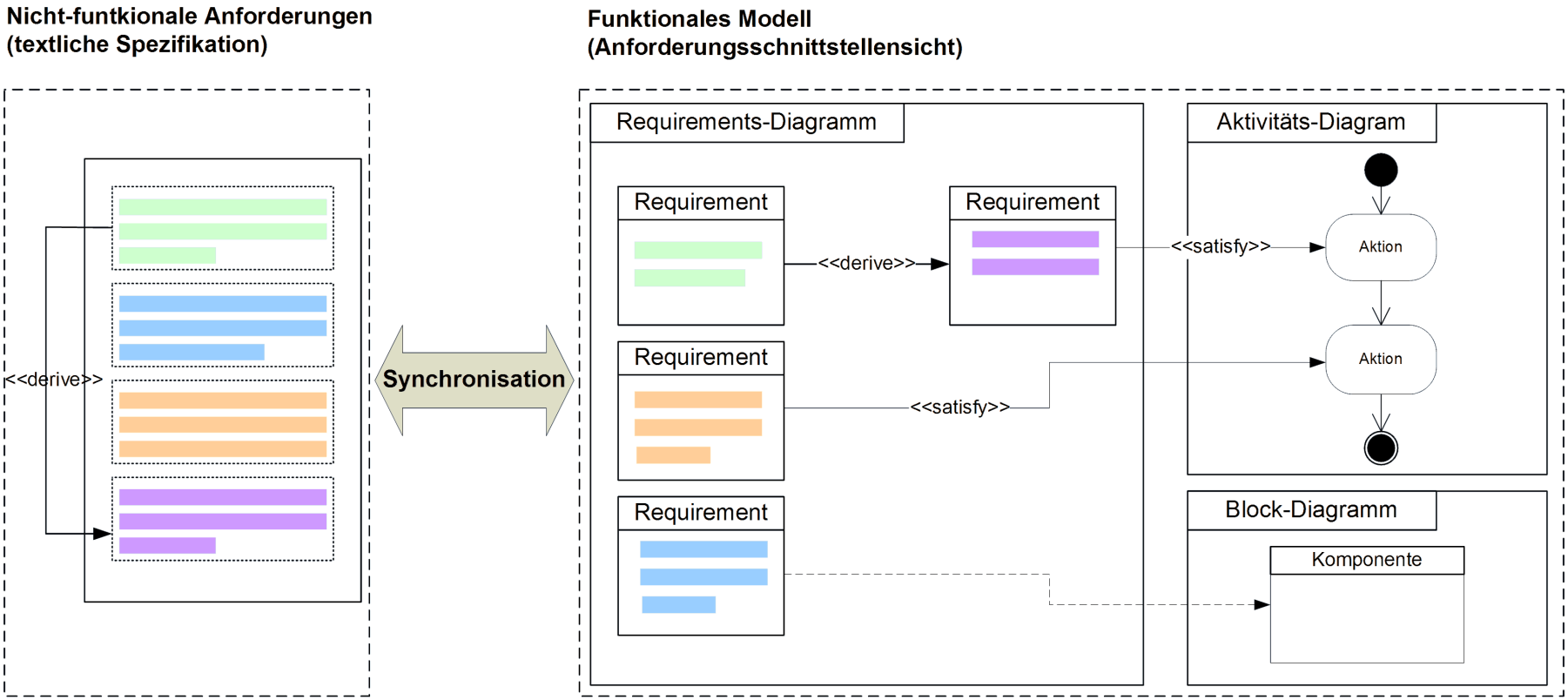


Abbildung 5.29.: Prinzip der Verknüpfung textlicher Anforderungen mit dem funktionalen Modell

Requirements und dem rechts gezeigten Aktivitätsdiagramm zu erkennen. So werden die Anforderungen aus externen Dokumenten durch `<<satisfy>>`-Beziehungen angekoppelt (um anzuzeigen, dass im Anforderungsmodell die externen Anforderungen berücksichtigt wurden) und die nicht-funktionalen Anforderungen über eine einfache Dependency-Relation an eine Komponente des funktionalen Modells angebunden (womit beschrieben wird, dass sich die nicht-funktionale Anforderung nur auf diese Komponente bezieht).

Prototypisch kann eine solche Werkzeugkette beispielsweise durch *IBM/telelogic Rhapsody Gateway* und *IBM/telelogic DOORS* zusammen mit *IBM/telelogic Rhapsody UML* realisiert werden. Ein anderes Produkt zur Umsetzung dieser Funktionalität ist *MagicRQ* für *MagicDraw UML* von *No Magic, Inc.*

Im funktionalen Modell wird die Schnittstelle zu den extern verwalteten textlichen Anforderungen dabei durch die *Anforderungsschnittstellensicht* repräsentiert, die in Abschnitt 5.1.4.6 näher beschrieben wurde. Diese besteht im Wesentlichen aus einer untereinander und mit den Komponenten des funktionalen Modells verknüpften Menge von SysML-Requirements, die außerhalb des funktionalen Modells verwaltet wird.

6. Inkrementell-iterativer Erstellungsprozess für Anforderungsspezifikationen

In diesem Hauptkapitel wird als weiteres wesentliches Element des Gesamtkonzepts zur modellbasierten Anforderungsspezifikation die Prozessdefinition erläutert. Durch den im Folgenden beschriebenen Prozessablauf lässt sich eine Anforderungsspezifikation konform mit dem beschriebenen SysML(A)-Metamodell erstellen (Kapitel 5).

Generell strukturieren Prozessmodelle die Bearbeitung von Aufgabenstellungen in überschaubare Zeit- und Informationsmengen. Sie legen fest, welche Informationen wann, wie und ggf. durch wen bearbeitet werden und welche Ergebnisse jeder Bearbeitungsschritt erzielen soll. Bei sicherheitskritischen Systemen umfassen Vorgehensmodelle zusätzlich Konstrukte zur Sicherstellung von Fehlerfreiheit und Widerspruchsfreiheit. Im Bereich der Software- und Systementwicklung sind Vorgehensmodelle - neben Methoden und Werkzeugen - längst ein etabliertes Element des BMW-Konzepts. Wie bereits in Abschnitt 4 beschrieben, erscheint jedoch ebenso sinnvoll, auch die Erstellung von Anforderungsspezifikationen einem Prozessmodell zu unterwerfen. Dies gilt insbesondere, wenn, wie im hier beschriebenen Ansatz, ähnliche Beschreibungsmittel (UML, SysML) wie bei der Systementwicklung eingesetzt werden. In den folgenden Unterkapiteln soll daher ein Prozessmodell entwickelt werden, das speziell auf die Erstellung von Anforderungsspezifikationen ausgerichtet ist. Das Prozessmodell berücksichtigt dabei insbesondere die Erstellung des funktionalen Modells (siehe Abschnitt 5.1), enthält aber auch Elemente zur Organisation nicht-funktionaler Anforderungen und deren Verknüpfung zum funktionalen Modell.

In Abschnitt 6.1 werden zunächst die Anforderungen an den Prozess formuliert und eine geeignete Klasse von Vorgehensmodellen ausgewählt. Darauf basierend wird ein Prozessmodell erstellt, dessen Eigenschaften in Abschnitt 6.2 beschrieben werden. Als wesentliche Elemente des Prozesses werden in Abschnitt 6.3 die einzelnen Phasen und in Abschnitt 6.4 die einzelnen Subprozesse detailliert vorgestellt. Bezogen auf das BMW-basierte Gesamtkonzept behandelt dieses Kapitel den in Abbildung 6.1 grau hervorgehobenen Bereich.

Weiterhin wird in Abschnitt 6.5 eine Dokumentationsmethodik beschrieben, mit der das funktionale Anforderungsmodell - als Ergebnis des Prozessablaufs - langfristig archiviert werden kann. Damit entsteht ein Endprodukt, das möglichst unabhängig von der Weiterentwicklung und Verfügbarkeit der eingesetzten Modellierungswerkzeuge ist.

6.1. Wahl des Vorgehensmodells

Zum besseren Verständnis der nachfolgenden Aussagen sollen zunächst einige Begriffe für die Verwendung im Rahmen dieser Arbeit definiert werden. Ziel ist die Definition eines *Prozesses* zur Erstellung eines Anforderungsmodells entsprechend dessen Definition in Kapitel 5. Dieser Prozess wird durch ein *Prozessmodell* definiert. Basis für das konkrete Prozessmodell ist ein allgemeines *Vorgehensmodell*, das die grundlegenden Eigenschaften des Prozessmodells festlegt.

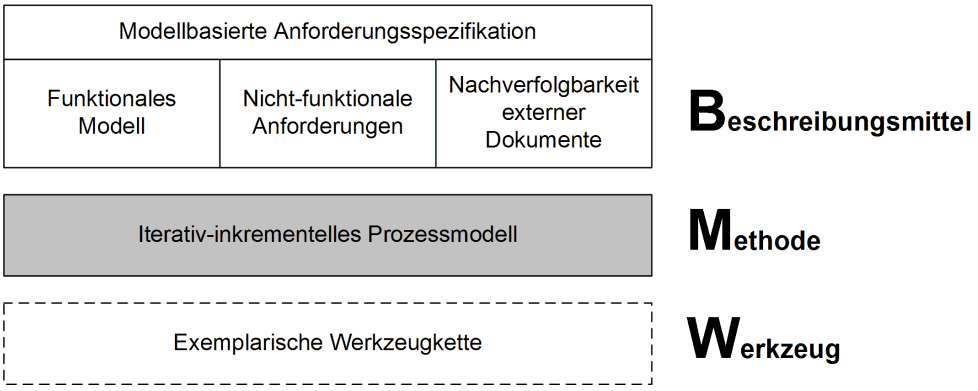


Abbildung 6.1.: Einordnung des Prozesses in den BMW-Kontext

Dieses Grundgerüst wird dann entsprechend der speziellen Gegebenheiten der Anforderungsmodellierung ausgebaut und zum Prozessmodell verfeinert.

In Kapitel 6.1.1 werden zunächst die Anforderungen an das Vorgehensmodell definiert. Auf Basis dieser Anforderungen wird im nachfolgenden Abschnitt 6.1.2 dann ein geeignetes Vorgehensmodell ausgewählt.

6.1.1. Anforderungen an das Vorgehensmodell

Um ein geeignetes Vorgehensmodell ableiten zu können, sollen in diesem Unterabschnitt zunächst die spezifischen Anforderungen ermittelt werden. Dabei müssen einige Besonderheiten berücksichtigt werden, die typisch für die Modellierung von Anforderungen sind und in Software-Entwicklungsprozessen nicht oder nicht in der gleichen Form auftreten.

Die auffälligste Besonderheit bei der Erstellung einer Anforderungsspezifikation ist die Tatsache, dass die Arbeiten mit einer minimalen Menge an Ausgangsinformationen beginnen. Üblicherweise sind dies grobe Skizzen oder stichpunktartige Notizen über das zu spezifizierende System. Anders ist dies bei der System- oder Softwareentwicklung: Hier ist der Ausgangspunkt ein Lasten- oder Pflichtenheft, dass bereits sehr viele Informationen über das zu entwickelnde System enthält. Während Prozesse zur Systementwicklung also im Wesentlichen Informationen transformieren, muss der hier verwendete Prozess die Sammlung und Herausarbeitung von Informationen beherrschen. Er muss daher mit einer geringen Ausgangsmenge an Informationen auskommen und diese systematisch, Schritt für Schritt um weitere Artefakte ergänzen können. Das zugrunde liegende Prozessmodell muss diese Arbeitsweise unterstützen.

Weiterhin ist die Phase der Anforderungserstellung durch ein hohes Maß an Interaktion zwischen den Projektbeteiligten geprägt, die während der Erstellung der Anforderungsspezifikation zu vielen Modifikationen und Änderungen führen kann. Oftmals werden auch verschiedene Varianten einer Problemstellung geprüft, die dann auf ihre Tauglichkeit untersucht und entsprechend angenommen oder verworfen werden müssen. Auf diese zahlreichen und häufigen Anpassungen muss das gewählte Prozessmodell mit einer hohen Flexibilität reagieren können.

Ebenfalls muss bei der Wahl des Vorgehensmodells beachtet werden, dass der hier verfolgte modellbasierte Ansatz ideal geeignet ist, um durch ausführbare Modelle das Verhalten des zukünftigen Systems bereits in frühen Phasen anschaulich darzustellen. Die frühe Ausführbarkeit ermöglicht es zudem, bereits Teile des Anforderungsmodells testen und gegebenenfalls verifizieren zu können. Dieser Vorteil der ausführbaren Modelle lässt sich allerdings nur dann ausnut-

zen, wenn die Arbeitsschritte durch den Prozess so gesteuert werden, dass möglichst schnell ein hinreichender Modell-Reifegrad erreicht wird.

Ein geeignetes Vorgehensmodell muss somit:

- eine schrittweise Anreicherung des Modells mit Informationen ermöglichen
- schnell und flexibel auf Modifikationen reagieren können
- möglichst früh eine Modellausführung ermöglichen

Diese Anforderungen grenzen die Menge möglicher Vorgehensmodelle ein und legen den Fokus auf iterativ-inkrementelle, flexible Methodiken.

6.1.2. Auswahl des Vorgehensmodells

Basierend auf den im vorherigen Abschnitt beschriebenen Anforderungen sollen im Folgenden verschiedene Arten von Vorgehensmodellen vorgestellt und auf ihre Eigenschaften hin untersucht werden. Das nahe liegendste Vorgehensmodell für sicherheitskritische Systeme und insbesondere für Systeme im Bahnbereich ist das V-Modell [KBS06], das in der DIN EN 50126 [EN50126, S. 26] für die Beschreibung des Lebenszyklus eines Systems verwendet wird. Daher liegt es nahe, dieses Vorgehensmodell auch für die Erstellung der vorgelagerten Anforderungsspezifikation zu verwenden. Das V-Modell basiert auf der schrittweisen Verfeinerung eines Entwurfs von den groben Systemanforderungen bis hin zum Coding der Software und dem ebenso schrittweisen Validieren des Systems vom Code über die einzelnen Module bis hin zum Gesamtsystem. In der Urform, in der die beiden Schenkel des V jeweils nur einmal für das gesamte zu entwickelnde System durchlaufen werden, ähnelt das V-Modell dem Wasserfallmodell [ROY70] und übernimmt damit auch dessen Probleme:

- Der gesamte Funktionsumfang muss auf einmal implementiert werden
- Der Prozess ist wegen seiner Linearität unflexibel, da Änderungen in Artefakten einer späteren Phase nicht einfach auf Artefakte früherer Phasen zurückübertragen werden können.
- Ausführen und Testen der erstellten Artefakte ist erst in sehr späten Phasen möglich
- Fehler werden somit sehr spät erkannt

Bereits der erste Punkt disqualifiziert dieses Vorgehensmodell für die Erstellung von Anforderungsspezifikationen, da dort naturgemäß niemals alle Informationen von Anfang an und auf einmal verarbeitet werden können.

Aus den genannten Gründen wird dieses Vorgehensmodell auch bei der Systementwicklung kritisch gesehen. Dies führte zur Entstehung neuer Ansätze, wie beispielsweise dem Spiralmodell von Boehm [BOE88], das eine inkrementelle Entwicklung eines Systems erlaubt. Dabei werden die Entwicklungsphasen nicht einmal für das Gesamtsystem, sondern mehrfach für kleine Untereinheiten des Gesamtsystems durchlaufen. Nach jedem Durchlauf wird das Ergebnis evaluiert und daraus Schlüsse für die Bearbeitung des nächsten Inkrements gezogen. Wie Cockburn in [COC93] beschreibt, müssen sich Spiralmodell und V-Modell dabei nicht zwangsweise ausschließen. Vielmehr kann ein inkrementeller Prozess als eine Abfolge vieler „kleiner" V-Prozesse verstanden werden, die pro Inkrement jeweils einen Teil des Systems implementieren.

Aus dem grundlegenden Spiralmodell wurden mittlerweile zahlreiche *iterativ-inkrementelle* Software- und Systementwicklungsprozesse abgeleitet. Diese stellen die einzelnen Spiraldurchläufe als wiederholte, lineare Durchläufe durch die gleichen Prozessschritte dar. Beispiele dafür sind der Rational Unified Process (RUP) [RUP98], der Object Engineering Process (OEP) [OSK06]

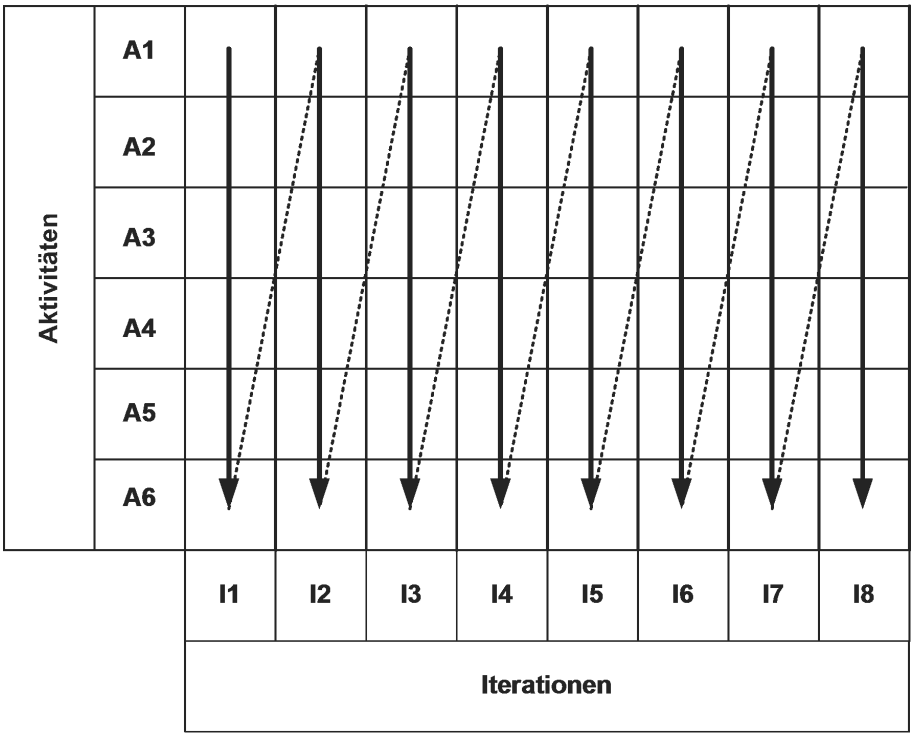


Abbildung 6.2.: Allgemeiner iterativ-inkrementeller Prozessablauf

und der OPRAIL-Prozess [OPR06]. Gemeinsam ist diesen Prozessen, dass sie eine Gesamtaufgabenstellung (z.B. die Umsetzung eines Pflichtenhefts in ein fertiges Produkt) in kleine Teilaufgaben zerlegen, für die jeweils alle nötigen Aktivitäten (z.B. Erstellung Systemdesign, Erstellung Softwareanforderungen, Erstellung Softwarearchitektur, Codeerstellung) iterativ durchlaufen werden. Nach Fertigstellung einer Teilaufgabe wird in der nachfolgenden Iteration die nächste Teilaufgabe dem System hinzugefügt, wodurch dieses inkrementell wächst. Abbildung 6.2 zeigt eine Prinzipdarstellung dieses Vorgehensmodells. Die einzelnen pro Iteration zu durchlaufenden Aktivitäten sind in dieser Darstellung zeilenweise angeordnet, die einzelnen Iterationen in Spalten. Die hervorgehobenen Pfeile zeigen an, in welcher Reihenfolge die einzelnen Aktivitäten pro Iteration durchlaufen werden sollen. Die gestrichelte Linie zeigt den Rücksprung in die erste Aktivität zu Beginn einer neuen Iteration.

Diese Art von Vorgehensmodellen ist wegen ihrer Dynamik und der schrittweisen Vervollständigung des Systems gut für die Erstellung von Anforderungsspezifikationen geeignet. Daher wird der im Rahmen dieser Arbeit genutzte Prozess auf dem beschriebenen iterativ-inkrementellen Vorgehensmodell aufgebaut und dessen prinzipielle Eigenschaften übernommen.

Ein deutlicher Unterschied zu Prozessen für die Systementwicklung ergibt sich jedoch durch die nur allmähliche Anreicherung des Modells mit Informationen: Während bei der Systementwicklung das Pflichtenheft von Anfang an vorliegt und inkrementell abgearbeitet werden kann, müssen bei der Anforderungserstellung die benötigten Informationen Schritt für Schritt „aus dem Nichts” erzeugt werden. Dies erfordert eine zeitliche Staffelung der einzelnen Aktivitäten im Prozess, was durch eine Berücksichtigung der einzelnen erforderlichen Aufgaben zur Erstellung der Anforderungsspezifikation erreicht wird. Diese Aufgabenorientierung wird in Abschnitt 6.2.3 beschrieben.

6.1.3. Einordnung in den Gesamtlebenszyklus

Die Erstellung der Anforderungsspezifikation ist ein elementarer Teil des gesamten Produktlebenszyklus. Daher muss sich auch der im Rahmen dieser Arbeit beschriebene Prozess in den Produktlebenszyklus integrieren. Weiterhin ist im Umfeld sicherheitskritischer bahntechnischer Systeme insbesondere die Ankopplung der Risikoanalyse und der Umgang mit deren Ergebnissen zu berücksichtigen. Die Berücksichtigung dieser beiden Aspekte im Prozessmodell wird in den folgenden beiden Unterkapiteln beschrieben.

6.1.3.1. Wechselwirkungen mit dem Lebenszyklus nach DIN EN 50126

Für Bahnanwendungen gilt die Spezifikation des Lebenszyklus gemäß DIN EN 50126 [EN50126], der in Bild 6.3 in Form des V-Modells dargestellt ist.

Prinzipiell sind die Abläufe innerhalb des hier beschriebenen Prozesses unabhängig von den einzelnen Phasen des Lebenszyklus nach DIN EN 50126. Allerdings existieren an diskreten Punkten Abhängigkeiten zwischen der Anforderungserstellung und dem im V-Modell beschriebenen Lebenszyklus, die den Reifegrad der modellbasierten Anforderungsspezifikation und ausgetauschte Informationen betreffen. In Abbildung 6.3 sind diese Abhängigkeiten rot hervorgehoben. Zu erkennen sind zwei Arten von Abhängigkeiten:

- Abhängigkeiten bezüglich des zeitlichen Ablaufs, dargestellt durch rote Quadrate: Innerhalb des Quadrates ist diejenige Prozess-Phase (siehe Abschnitt 6.3) aufgeführt, die mindestens erreicht sein muss, wenn die entsprechende Phase im Systemlebenszyklus bearbeitet wird. Beispielsweise muss für die Durchführung der Risikoanalyse mindestens Phase P1 bei der Anforderungsmodellierung abgeschlossen sein, weil für die Risikoanalyse eine Aufstellung der Systemfunktionen benötigt wird.
- Abhängigkeiten bezüglich der ausgetauschten Informationen: Die roten Pfeile geben an, in welchen Lebenszyklusphasen welche Informationen der modellbasierten Anforderungsspezifikation entnommen bzw. für ihre Erstellung zur Verfügung gestellt werden. Für die Lebenszyklusphase „Systemdefinition" werden zum Beispiel die in der Systemabgrenzungssicht enthaltenen Daten benötigt.

In Tabelle 6.1 sind die zwischen dem Lebenzyklusmodell und dem Anforderungsmodell ausgetauschten Informationen tabellarisch aufgeführt und genauer beschrieben.

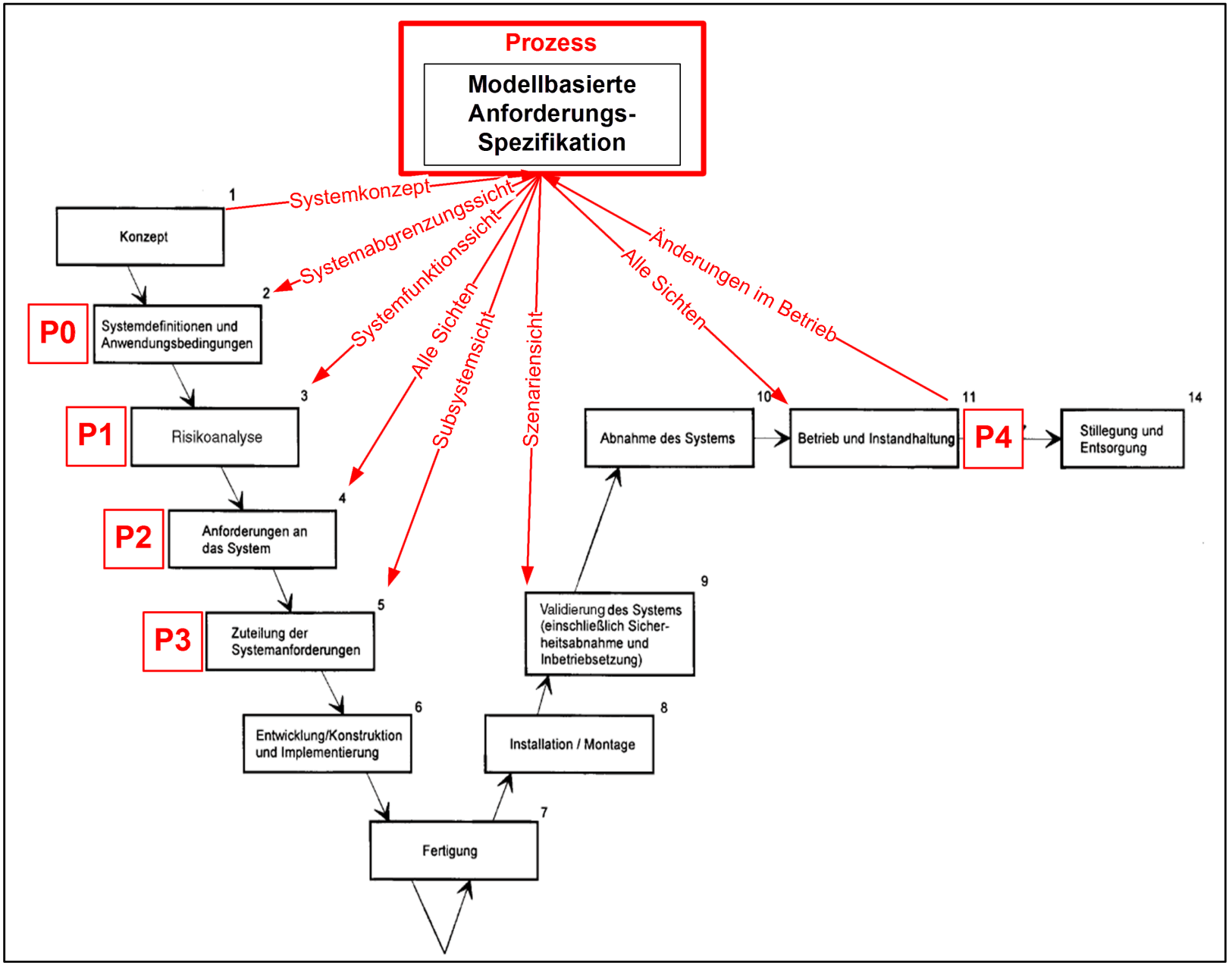


Abbildung 6.3.: Produktlebenszyklus gemäß DIN EN 50126 [EN50126, S. 26]

Von	Nach	Ausgetauschte Information
Lebenszyklusphase 1 (Konzeptphase)	modellbasierte Anforderungsspezifikation	Informationen zum Systemkonzept. Diese werden als Grundlage für die Erstellung des funktionalen Modells benötigt
modellbasierte Anforderungsspezifikation	Lebenszyklusphase 2 (Systemdefinition und Anwendungsbedingungen)	Die Systemabgrenzungssicht kann für die Systemdefinitionsphase genutzt werden, da sie zwingend eine Definition des Systems bereitstellt
modellbasierte Anforderungsspezifikation	Lebenszyklusphase 3 (Risikoanalyse)	Systemfunktionssicht. Wie in 5.1.4.2.1 beschrieben wurde, kann die Ableitung der einzelnen Systemfunktionen in der Systemfunktionssicht für die Risikoanalyse verwendet werden.
modellbasierte Anforderungsspezifikation	Lebenszyklusphase 4 (Anforderungen an das System)	Alle Sichten. Über diesen Informationsfluss wird die in der modellbasierten Anforderungsspezifikation repräsentierte Systemspezifikation in den Lebenszyklusprozess eingespeist.
modellbasierte Anforderungsspezifikation	Lebenszyklusphase 5 (Zuteilung der Systemanforderungen)	Subsystemsicht. Obwohl in der Nomenklatur der modellbasierten Anforderungsspezifikation die Subsystemsicht zu den Systemanforderungen gehört, wird durch diesen Informationsfluss deutlich gemacht, dass diese explizit auch der Phase 5 des Lebenszyklusmodells zur Verfügung gestellt wird.
modellbasierte Anforderungsspezifikation	Lebenszyklusphase 9 (Validierung)	Szenariensicht. Die einzelnen Szenarien können in dieser Phase als Systemtestfälle verwendet werden, um das System gegen die modellbasierte Anforderungsspezifikation zu validieren.
Lebenszyklusphase 11 (Betrieb und Instandhaltung)	modellbasierte Anforderungsspezifikation	Änderungen am System zur Laufzeit. Wird das reale System modifiziert, sollten diese Änderungen auch in der modellbasierten Anforderungsspezifikation nachgezogen werden. Dadurch wird die Konsistenz zwischen System und Modell sichergestellt.

Tabelle 6.1.: Austausch zwischen Lebenszyklusmodell und Prozess zur Anforderungserstellung

6.1.3.2. Einbindung der Risikoanalyse

Wie im vorherigen Kapitel bereits angeschnitten wurde, existieren als Untermenge der generellen Abhängigkeiten zwischen dem hier beschriebenen Prozessmodell und dem Lebenszyklus nach DIN EN 50126 auch Abhängigkeiten zwischen der Risikoanalyse und den einzelnen Phasen des Prozessmodells. Auf Grund der besonderen Bedeutung der Risikoanalyse für sicherheitskritische Systeme sollen diese speziellen Abhängigkeiten in diesem Unterkapitel gesondert dargestellt werden. Dabei soll herausgearbeitet werden:

(a) welche Eingangsdaten die Risikoanalyse aus dem Anforderungsmodell benötigt

(b) ab welchem Prozessschritt diese Daten verfügbar sind

(c) welche Daten aus der Risikoanalyse in das Anforderungsmodell zurückfließen

Wie in Abschnitt 5.1.4.2.1 beschrieben, sind die wesentlichen Eingangsparameter für die Risikoanalyse eine eindeutige Systemabgrenzung sowie eine Aufstellung aller Systemfunktionen. Für die Einordnung in den Lebenszyklus nach DIN EN 50126 bedeutet dies, dass das Anforderungsmodell einen hinreichenden Reifegrad insbesondere der Systemabgrenzungssicht, als auch der Systemfunktionssicht benötigt. Dies ist gemäß Abschnitt 6.3.2 frühestens nach Ende der Phase P1 der Fall. Daher kann mit der Durchführung der Risikoanalyse erst begonnen werden, wenn dieser Meilenstein bei der Erstellung des Anforderungsmodells erreicht wurde.

Die Durchführung der Risikoanalyse an sich erfolgt dann vollständig außerhalb der hier beschriebene Anforderungsmodellierung. Ergebnis der Risikoanalyse ist eine funktionsbezogene Menge an *ertragbaren Gefährdungsraten* (tolerable hazard rates, THR). Diese geben an, mit welcher Rate die jeweilige Systemfunktion sicherheitskritisch versagen darf, damit das Gesamtsystem den Risikogrenzwert nicht überschreitet. Die THRs sind wichtige Werte für die Sicherheitsanalyse bei den Herstellern und sollten direkt im Anforderungsmodell abrufbar sein. Da sich dabei verschiedenen Möglichkeiten ergeben - z.B. entweder direkt den Zahlenwert aus der Risikoanalyse oder eine SIL-Einstufung gemäß DIN EN 50129 zu verwenden - soll hier nur ein generischer Weg beschrieben werden.

Prinzipiell handelt es sich sowohl bei den THR-Zahlenwerten, als auch bei der SIL-Einstufung um nicht-funktionale Anforderungen, ähnlich z.B. Vorgabewerten für ertragbare Erschütterungen und EMV. Es wird daher für den Rahmen dieser Arbeit vorgeschlagen, die Ergebnisse der Risikoanalyse wie andere nicht-funktionale Anforderungen zu behandeln und gemäß den Ausführungen in Abschnitt 5.2 an das funktionale Modell anzukoppeln. Diese Ankopplung sollte dabei jeweils zwischen einem Blatt des Baums der Anwendungsfälle (siehe Abschnitt 5.1.4.2.3) und einem Requirement erfolgen, das für diese Systemfunktion die THR oder SIL-Einstufung enthält.

6.2. Ableitung des Prozessmodells

Aus dem allgemeinen iterativ-inkrementellen Vorgehensmodell wird in diesem Abschnitt das konkrete Prozessmodell abgeleitet. Dazu werden zunächst in Unterabschnitt 6.2.2 die wesentlichen Strukturmerkmale des Prozesses vorgestellt. Über die in Abschnitt 6.2.3 beschriebene Aufgabenorientierung erfolgt dann eine Anpassung des allgemeinen Vorgehensmodells an die besonderen Belange der Anforderungsmodellierung, die zum letztendlich verwendeten Prozessmodell führt. Zunächst wird aber im folgenden Abschnitt 6.2.1 beschrieben, wie in einer Vorlaufphase der Einstieg in die Anforderungsmodellierung vorbereitet werden kann.

6.2.1. Informeller Prozessvorlauf

Auch wenn es theoretisch möglich wäre, mit dem in diesem Abschnitt beschriebenen Prozessmodell ohne jede Vorarbeiten ein Anforderungsmodell zu erstellen, entspricht dies nicht dem üblichen Vorgehen bei der Entwicklung von eisenbahntechnischen Systemen. Gerade in der Fachdomäne des Eisenbahnwesens entstehen auch neue Systeme selten „auf der grünen Wiese". Vielmehr ist ein grobes Anforderungsprofil entweder von ähnlichen Vorgängersystemen bekannt oder wird während den frühesten Überlegungen zum neuen System entwickelt.

Ein solches Anforderungsprofil besteht üblicherweise aus natürlich-sprachlichen Festlegungen, informellen Skizzen oder aus Passagen bereits bestehender Lastenhefte und Dokumente. Ein gutes Beispiel für die Art dieser Informationen findet sich innerhalb dieser Arbeit selbst, in der Beschreibung der Bahnübergangssicherungsanlage (siehe Abschnitt 6.2.6), auf die das Beispielmodell angewendet wird. Auch dort werden natürlich-sprachliche Textpassagen und Projektierungs-Grafiken verwendet, um einen groben Überblick über das zu modellierende System zu erhalten.

Für die Durchgängigkeit der Anforderungsspezifikation und die Sicherstellung der vollständigen Umsetzung aller Anforderungen ist es daher wichtig, diese Informationen in geeigneter Weise mit dem eigentlichen Modellierungsprozess zu verbinden. Dazu müssen in einer Vorlaufphase vor dem eigentlichen Modellierungsprozesses informelle Anforderungen gesammelt und so verwaltet werden, dass sie auf Artefakte des späteren Modells verlinkt werden können. Durch diese Verlinkung kann dann gezeigt werden, dass alle Anforderungen aus der Vorlaufphase bei der Erstellung des Modells berücksichtigt wurden.

Es empfiehlt sich daher, die Informationssammlung der Vorlaufphase wie andere externe Dokumente und informelle Anforderungen gemäß den Ausführungen in Abschnitt `5.2.1` zu behandeln.

Damit ergeben sich einige Anforderungen an die Vorgehensweise, wie die Informationssammlung in der Vorphase zu erfolgen hat:

- Ermittlung aller Vorschriften und Verordnungen, die für das neu zu entwickelnde System relevant sind
- Auflistung aller Anforderungen, die z.B. in ersten Besprechungen zum neuen System ermittelt wurden, inklusive Skizzen und Zeichnungen
- Bereitstellung dieser Informationsmenge in einem einheitlichen Format für den Modellierungsprozess

Eine Möglichkeit, wie diese Datenmenge in geordneter Weise dem Modellierungsprozess zur Verfügung gestellt werden kann, liegt in der Benutzung eines Anforderungsmanagement-Werkzeuges wie z.B. IBM/telelogic DOORS. Setzt man ein solches Werkzeug bereits von Anfang an ein bzw. überträgt andere Informationsquellen in die entsprechende Datenbank, so gelten die in Abschnitt 5.2.3 gemachten Aussagen auch für die vor dem eigentlichen Prozessbeginn gesammelten Informationen. Damit stünde während der Laufzeit des Modellierungsprozesses eine einheitlich Datenbasis textlicher und informeller Anforderungen bereit, gegen die das funktionale Anforderungsmodell verlinken lässt.

6.2.2. Prozessstrukturierung

Zur Gliederung des Prozesses werden vier Elemente genutzt. Diese sind

- Phasen
- Iterationen
- Subprozesse
- Aktivitäten

Phasen repräsentieren die groben zeitlichen Abschnitte innerhalb der Erstellung des funktionalen Modells, wobei die im Prozessmodell definierten Phasen in 6.3 beschrieben sind.

Die Phasen fassen die Anwendung einzelner *Subprozesse* zusammen. Jeder Subprozess ist dabei einer der in Abschnitt 5.1.3.2 beschriebenen Sichten zugeordnet und erstellt bzw. überarbeitet

die entsprechenden Modellartefakte. Innerhalb einer Phase werden die Subprozesse in mehreren *Iterationen* durchlaufen. Eine Beschreibung der verwendeten Subprozesse findet sich in Abschnitt 6.4.

Jeder Subprozess besteht aus einzelnen *Aktivitäten*. Typischerweise sind die Aktivitäten so ausgestaltet, dass jede Aktivität eins der Inhaltselemente des Modells (siehe 5.1.2) modifiziert. Die einzelnen Aktivitäten werden in jedem Subprozess einmal in einer festgelegten Reihenfolge durchlaufen.

Diese grundlegende Struktur des Prozesses entspricht weitgehend dem OPRAIL-Prozessmodell [OPR06, S. 50], ist aber gegenüber diesem vereinfacht. So entfallen die Prozesselemente *Role*, *Person* und *Supporter*, die die Rollenverteilung bei der Entwicklung sicherheitskritischer Software repräsentieren und die durch die DIN EN 50129 vorgegeben sind [EN50129, S. 24]. Sie sind jedoch bei der Erstellung von Anforderungsmodellen nicht erforderlich und können daher entfallen. Bild 6.4 zeigt die Struktur und die Abhängigkeiten der einzelnen Prozesselemente als SysML-Blockdiagramm.

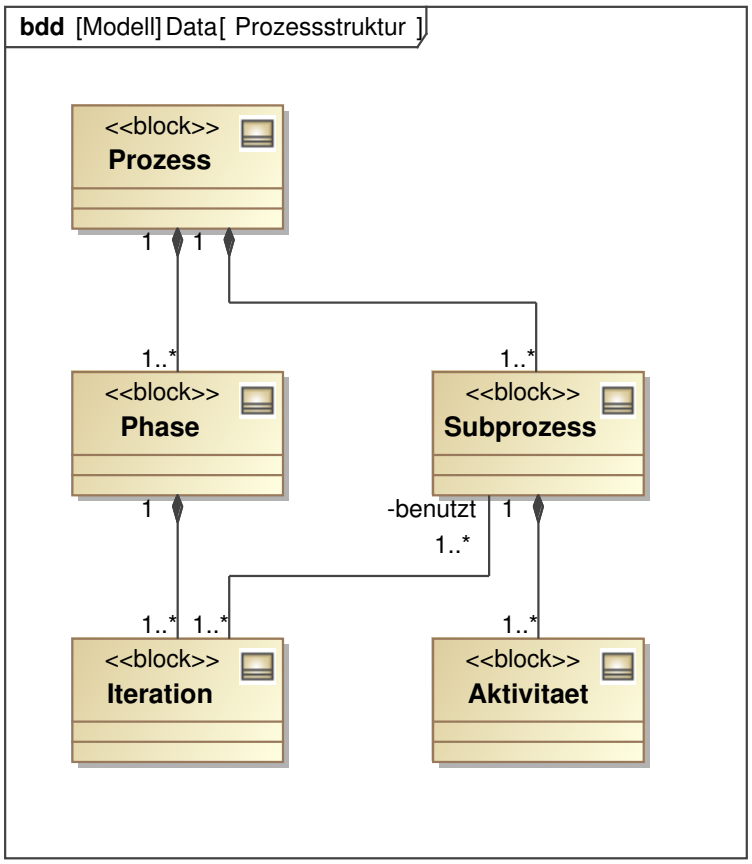


Abbildung 6.4.: Prinzipielle Struktur des Prozesses

6.2.3. Aufgabenorientierung

Wie in 6.1.2 ausgeführt, lässt sich das iterativ-inkrementelle Prozessmodell nicht direkt auf die Erstellung von Anforderungsspezifikationen übertragen. Zu Beginn des Prozesses liegt noch keine geschlossene, vollständige Aufgabenstellung vor, die iterativ-inkrementell abgearbeitet wer-

den könnte. Vielmehr soll diese ja erst innerhalb des Prozesses erstellt werden. Daher muss eine Methode gefunden werden, die den Informationsgehalt des Modells nach und nach erweitert, damit die Anforderungsspezifikation allmählich aufbaut und dabei die Grundlagen iterativ-inkrementeller Prozessmodelle berücksichtigt. Dies erfolgt in dieser Arbeit durch eine aufeinander aufbauende Menge von *Aufgaben*. Die Anzahl und der Umfang dieser Aufgaben orientiert sich dabei an den Sichten, die das Anforderungsmodell enthalten soll (siehe Abschnitt 5.1.3.2). Ziel jeder Aufgabe ist es, die zu den korrespondierenden Sichten gehörenden Modellelemente zu erstellen. Für das Anforderungsmodell ergeben sich aus den definierten Sichten die folgenden Aufgaben:

(a) Systemkontext spezifizieren ⇒erstellt die Systemabgrenzungssicht

(b) Systemfunktionen spezifizieren ⇒erstellt die Systemfunktionssicht und die Szenariensicht

(c) Systemverhalten spezifizieren ⇒erstellt die Systemverhaltenssicht

(d) Subsystemarchitektur spezifizieren ⇒ erstellt die Subsystemsicht

(e) Textliche Anforderungen sammeln und integrieren ⇒ erstellt die Anforderungsschnittstellensicht

Mit zunehmendem Reifegrad des Modells werden nacheinander die oben aufgeführten Aufgaben aktiviert. Jede aktive Aufgabe wird im Prozess durch die Ausführung zugehöriger Subprozesse repräsentiert. Eine neue Aufgabe kann immer erst dann aktiv werden, wenn das Modell für deren Bearbeitung genug Informationen enthält. So müssen beispielsweise die Systemfunktionen hinreichend genau definiert sein, bevor mit der Spezifikation des Systemverhaltens begonnen werden kann. Beachtet man alle Abhängigkeiten, ergibt sich die in der obigen Liste gezeigte Reihenfolge der Aufgaben. Einmal aktivierte Aufgaben bleiben über die gesamte Prozesslaufzeit aktiv und werden erst beendet, wenn der Erstellungsprozess des Anforderungsmodells insgesamt abgeschlossen wurde. Die gestaffelte Aktivierung von einzelnen Modellierungsaufgaben zeigt Abbildung 6.5.

Die über die Zeit stattfindende Kumulierung von aktiven Subprozessen ist erforderlich, weil die Aktivitäten in später begonnenen Subprozessen Änderungen und Ergänzungen an bereits zuvor erstellten Artefakten erforderlich machen können. Wird beispielsweise bei der Spezifikation des Systemverhaltens festgestellt, dass ein weiterer Akteur erforderlich ist, so muss dieser auch in die Systemabgrenzungssicht und in die Systemfunktionssicht aufgenommen werden. Diese Sichten können aber nur durch die zugehörigen Subprozesse modifiziert werden. Dieses Vorgehen nimmt den Gedanken der iterativ-inkrementellen Entwicklungsprozesse auf, in denen grundsätzlich *alle* Subprozesse wiederholt nacheinander durchgeführt werden. Es berücksichtigt aber, dass bei der Erstellung von Anforderungsmodellen auf Grund zunächst mangelnden Informationsgehalts die einzelnen Aktivitäten unterschiedlich gestaffelt begonnen werden müssen. In der Phase P0 besteht ein Iterationsdurchlauf lediglich aus den Aktivitäten des ersten Subprozesses, in der Phase P1 aus den Aktivitäten der ersten beiden Subprozesse, in der Phase P2 schließlich aus den Aktivitäten von drei Subprozessen, usw. Der beschriebene Ansatz passt somit den generischen inkrementell-iterativen Prozess den Belangen der Anforderungsmodellierung an. Eine Ausnahme ergibt sich dabei für die Anforderungsschnittstellensicht: Zum einen startet die zugehörige Aufgabe schon vor Beginn der eigentlichen Modellierungsarbeiten in einer Vorphase, um - wie in Abschnitt 6.2.1 beschrieben - die Integration bereits existierender, informeller Anforderungen sicherzustellen. Zum anderen können bei jeder der anderen Aufgaben Verknüpfungen zu externen Anforderungen erforderlich werden. Daher müssen die Verknüpfungspunkte zu den externen, textbasierten Anforderungsdokumenten bereits mit Beginn des Prozesses zur Verfügung gestellt werden. Aus diesem Grund durchläuft die Aufgabe „Textliche Anforderungen sammeln und integrieren" den gesamten Prozess.

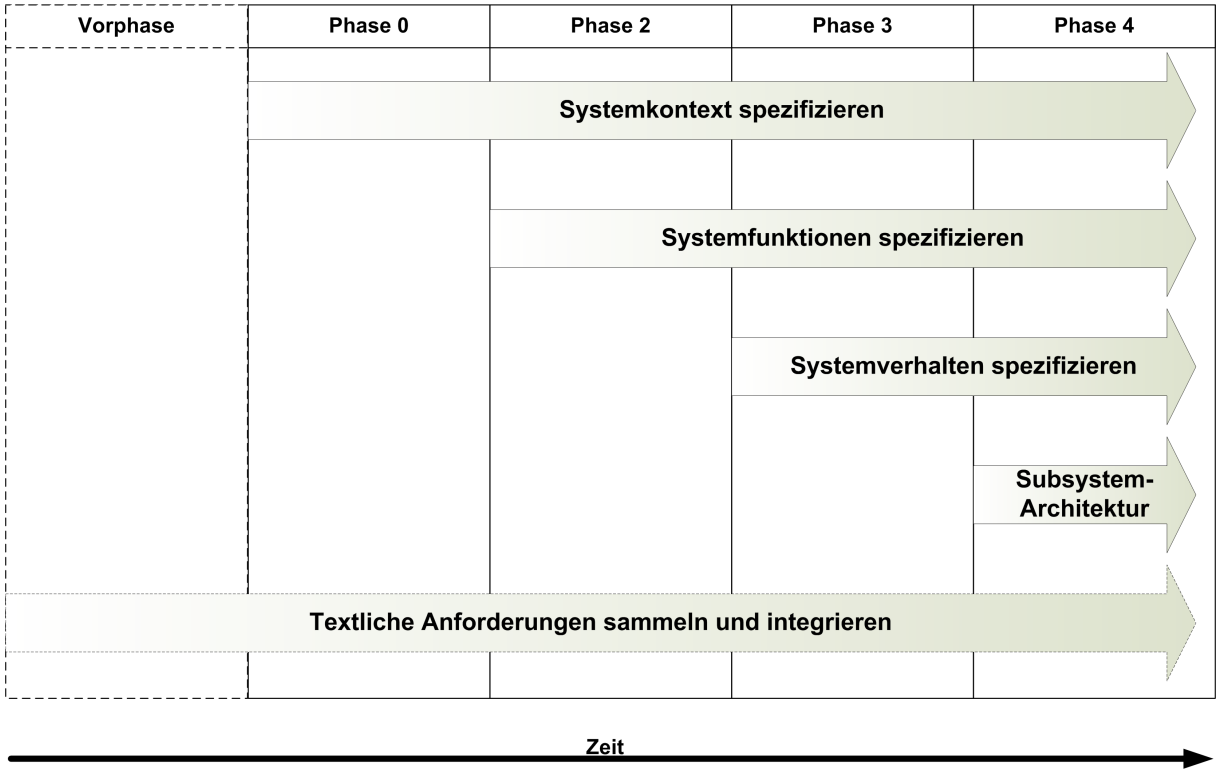


Abbildung 6.5.: Gestaffelte Aktivierung der Aufgaben im Prozess

Die informelle Vorphase wird in den folgenden Ausführungen zum Prozessmodell jedoch nicht näher betrachtet, da sie keine direkten Auswirkungen auf die Modellerstellung an sich hat.

Allgemein ist der Einstieg in den Prozess und die Definition der ersten Phase schwierig, da man gewissermaßen „auf einem leeren Blatt Papier" mit der Modellierung beginnen muss. Hierbei sind zwei verschiedene prinzipielle Vorgehensweisen denkbar: Zum einen könnte mit der Definition der Systemfunktionen begonnen werden, zum anderen mit der Spezifikation der Systemabgrenzung und des Systemkontextes. Für beide Vorgehensweisen lassen sich nachvollziehbare Argumente darlegen:

- Für die Funktionsspezifikation als ersten Schritt spricht, dass das Bedürfnis für ein neues oder überarbeitetes System meist aus der Änderung funktionaler Randbedingungen entsteht. Beispielsweise macht es die Migration zum ETCS (European Train Control System) erforderlich, dass in elektronischen Stellwerken die Möglichkeit zur Ankopplung an ein RBC (Radio Block Center) als neue Funktion vorgesehen wird. Hierbei sind üblicherweise einige der Systemfunktionen bereits bekannt und können als Ausgangspunkt für die Anforderungsmodellierung dienen.
- Für die Spezifikation des Systemkontextes als Ausgangspunkt lässt sich das Argument anführen, dass bei Aufgabenstellungen im Eisenbahnwesen oft eine starke Strukturorientierung vorliegt. Im Verbund der einzelnen Komponenten ist die Umgebung eines einzelnen Systems meist bekannt und ändert sich üblicherweise lediglich in längerfristigen Zyklen. Die Beschreibung der Systemumgebung und die Abgrenzung eines neuen Systems gegen seine Umwelt fällt den Modellierern daher meist leicht und entspricht der ingenieursmäßigen Denk- und Erfahrungswelt.

Letztendlich lässt sich keine eindeutig kausale Begründung für die Bevorzugung einer der beiden Möglichkeiten anführen. Zudem erlaubt der iterativ-inkrementelle Ansatz bei der Wahl kleiner Iterationen und einer kurzen ersten Phase eine quasi-parallele Abarbeitung beider Aufgaben. Für den hier vorgestellten Prozess wurde entschieden, mit der Spezifikation des Systemkontextes zu beginnen.

Die folgende grafische Darstellung (Bild 6.6 auf der nächsten Seite) zeigt den sich aus der Aufgabenorientierung ergebenden prinzipiellen Prozessablauf in Analogie zum Schaubild für das allgemeine iterativ-inkrementelle Vorgehensmodell (siehe Bild 6.2 auf Seite 92). Zu erkennen sind die Subprozesse als Kapselung der einzelnen Aktivitäten, die Phasen als zeitliches Gliederungselement und der iterative Prozessdurchlauf, wobei der Iterationsumfang bedingt durch die Aufgabenorientierung von Phase zu Phase anwächst. Dunkelgrau sind diejenigen Subprozesse markiert, die innerhalb einer Phase zwingend durchgeführt werden müssen. Die Ausführung der hellgrau markierten Subprozesse ist davon abhängig, ob diese zur Sicherstellung der Konsistenz im Modell erforderlich sind. Als eingängige Kurzformel lässt sich festhalten: „Neues" entsteht in den dunkelgrau markierten Subprozessen, in hellgrau markierten Subprozessen wird der Rest des Modells „nachgezogen".

6.2.4. rekursive Prozessanwendung

Zur Strukturierung des funktionalen Modells wurde in Kapitel 5.1.3.1 die *Ebene* eingeführt. Ebenen gliedern das Modell entlang eines zunehmenden Detaillierungsgrades und umfassen ein bis mehrere *Teilmodelle.* Zur Erstellung jedes Teilmodells wird der in diesem Kapitel beschriebene Prozess einmal durchlaufen. Ein Ergebnis der Prozessanwendung ist die Subsystem-Architektur, die Ausgangspunkt für die Anwendung des Prozesses auf der nächst tieferen Ebene ist: jede Subsystemkomponente der übergeordneten Ebene kann zum SuB der tieferen Ebene werden. Die Grafik in Bild 6.7 illustriert dieses prinzipielle Vorgehen.

In der Ebene 0 wird der Prozess auf das zu spezifizierende Gesamtsystem angewendet. Es entsteht das schwarz umrandete Teilmodell, das - neben anderen Sichten - auch die Systemabgrenzungssicht und die Subsystem-Architektur enthält.

Diese definiert die einzelnen Komponenten, aus denen sich das SuB in der nächst detaillierteren Betrachtungsebene zusammensetzt. Im dargestellten Beispielfall besteht das SuB der Ebene 0 aus den drei Subsystem-Komponenten *Subsystem1* (blau), *Subsystem2* (grün) und *Subsystem3* (rot), die zu einer Subsystemarchitektur verbunden sind. Sollen nun die einzelnen Subsysteme weiter spezifiziert werden, kann der Prozess rekursiv angewendet werden.

Dazu wird zunächst festgelegt, für welche der drei Subsysteme eine genauere Spezifikation nötig ist. Handelt es sich bei einem der Subsysteme beispielsweise um eine fertig zugekaufte Komponente eines Drittherstellers, so kann sie möglicherweise entfallen. Im gezeigten Beispiel wird jedoch jede der drei Subsystemkomponenten ausspezifiziert. Dazu wird für jedes Subsystem ein eigenes Teilmodell angelegt. SuB des jeweiligen Teilmodells ist dabei die entsprechende Subsystemkomponente aus Ebene 0. Diese Verbindung wird beispielhaft am *Subsystem3* dargestellt: Diese Subsystemkomponente aus Ebene 0 wird zum SuB in Ebene 1. Die anderen Subsystemkomponenten - *Subsystem1* und *Subsystem2* - werden zu Akteuren in deren Systemumfeld.

Auf jedes dieser Teilmodelle wird nun wieder der hier beschriebene Prozess angewendet. Dieser erzeugt je nach Erfordernis wiederum eine Subsystemarchitektur, die als Ausgangspunkt für eine noch detailliertere Spezifikation in Ebene 2 dienen kann. Das Gesamtmodell setzt sich letztendlich aus einer baumartigen Hierarchie einzelner Teilmodelle auf verschiedenen Ebenen

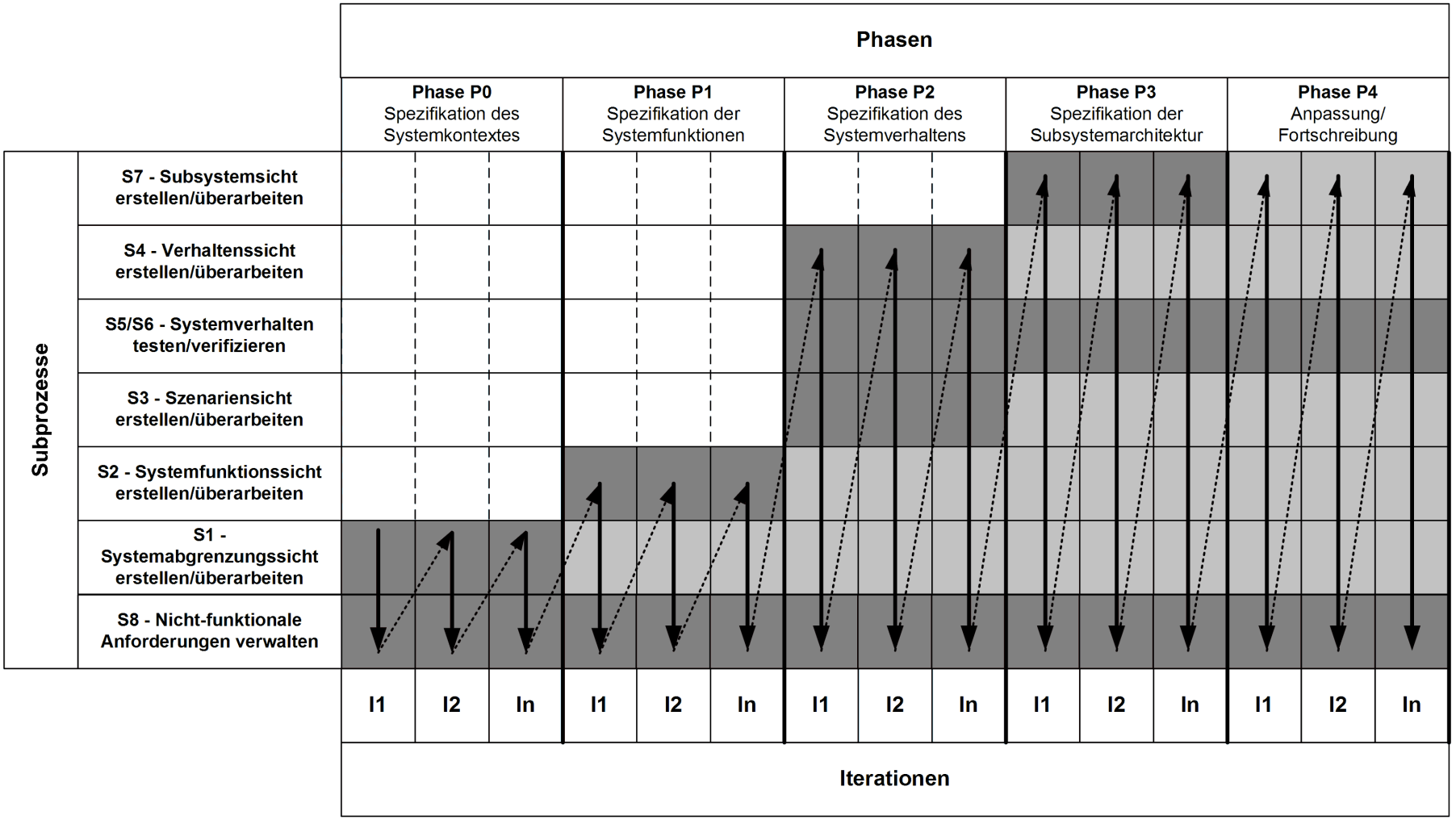


Abbildung 6.6.: Prinzipielle Darstellung des Prozesses

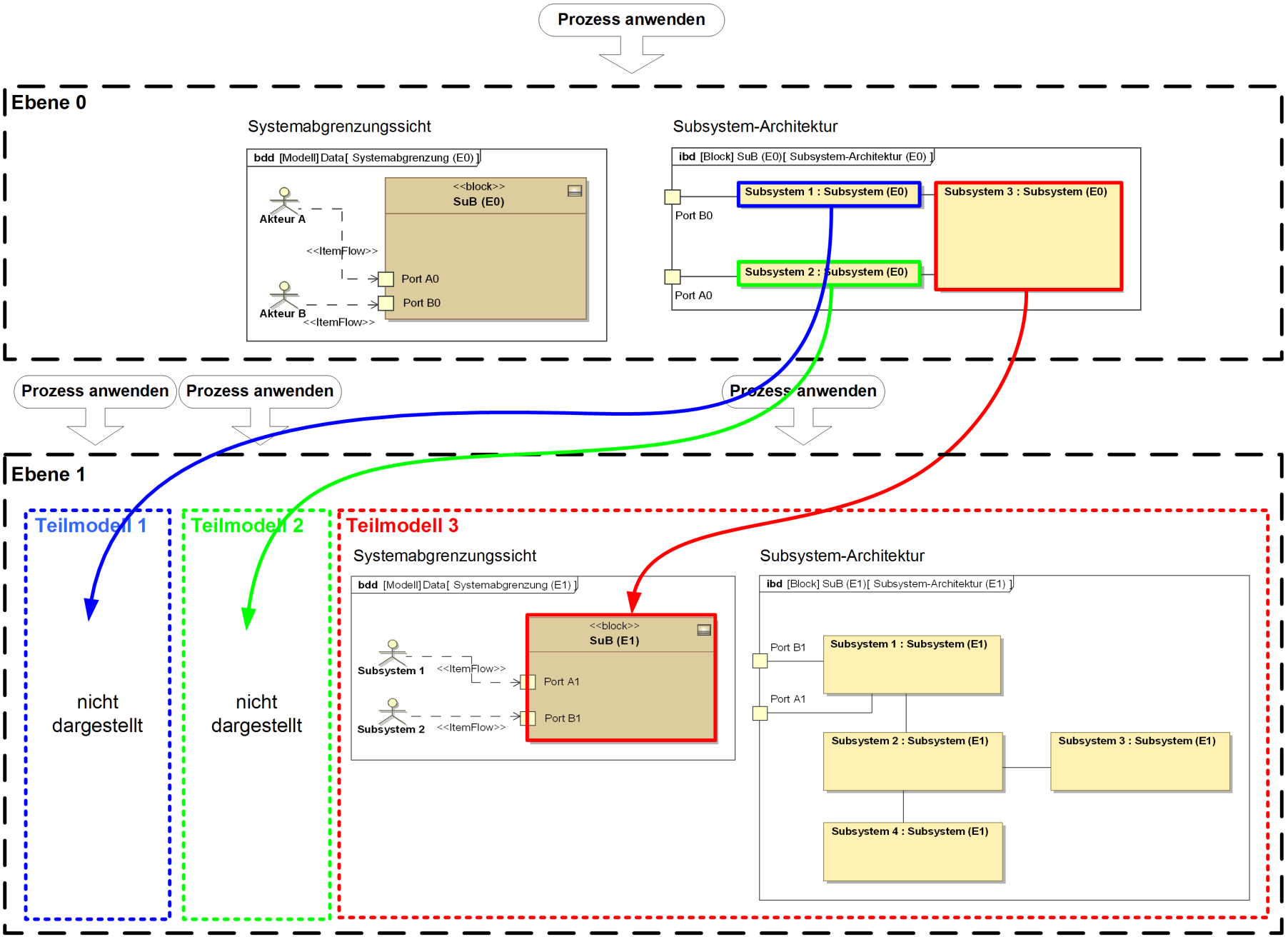


Abbildung 6.7.: Rekursive Anwendung des Prozesses

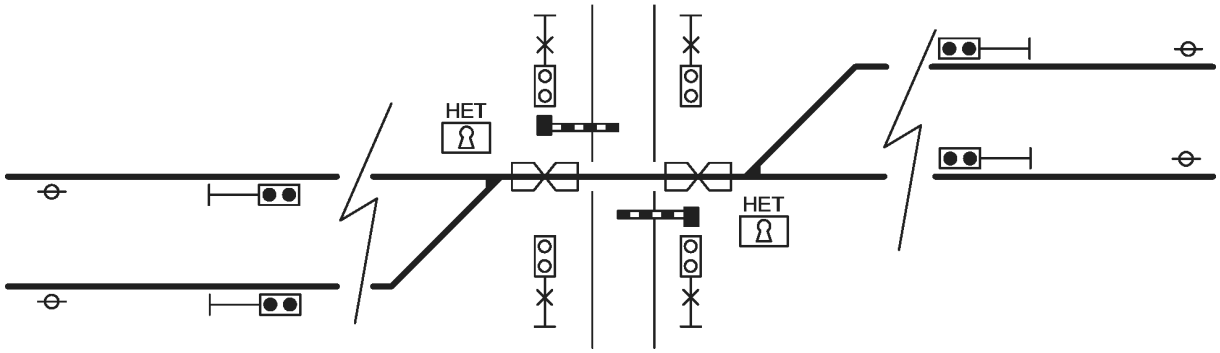


Abbildung 6.8.: Lageplanskizze BÜ

zusammen. Da in jeder Ebene individuell entschieden werden kann, ob und ggf. welche Subsystemkomponenten weiter ausspezifiziert werden, kann der Detaillierungsgrad des Gesamtmodells sehr genau gesteuert werden.

6.2.5. Prozessmetamodell

Der Prozessablauf ist in einem SysML-Modell beschrieben, das ähnlich dem strukturellen Metamodell (siehe Abschnitt 5.1.2) eine Vorgabe für eine konkrete Anwendung des Prozesses darstellt. Im Wesentlichen besteht dieses Modell aus mehreren, ineinander verschachtelten Aktivitätsdiagrammen, die die Abfolge der Phasen, Subprozesse und Aktivitäten definieren. In den folgenden Abschnitten zu Phasen und Subprozessen werden einzelne Diagramme aus dem Metamodell zur Veranschaulichung der Abläufe benutzt.

6.2.6. Beispielmodell

Das Vorgehensmodell wurde an einem durchgängigen Beispielmodell erprobt. Es handelt sich dabei um ein Anforderungsmodell für eine fiktive Bahnübergangssteuerung. Diese wurde gegenüber einem entsprechenden realen System modifiziert und lässt sich durch die folgenden Eigenschaften charakterisieren:

- Eingleisige Strecke im Kreuzungsbereich Eisenbahn/Straße
- Maximal 4 zu- bzw. ablaufende Streckenäste
- Einschaltung über Radsensoren, Belegtmeldung/Ausschaltung über Induktionsschleifen
- Hilfseinschalttasten (HET) als Schlüsselschalter
- Sicherung mit Lichtzeichen und Halbschranken
- Überwachung durch Überwachungssignal
- Zusätzlich Meldung des Betriebszustandes an eine Betriebsleitzentrale
- Diagnose-, Konfigurations- und Wartungsschnittstelle vor Ort

Eine Lageplanskizze eines dieser Beschreibung entsprechenden BÜ zeigt Abbildung 6.8.

Auf Basis dieser Vorgaben wurde ein mit dem Struktur-Metamodell (siehe 5.1.2) konformes, funktional orientiertes Anforderungsmodell erstellt, in dem alle im folgenden beschriebenen Prozessschritte durchgeführt wurden. Als Modellierungswerkzeug kam MagicDraw UML von

NoMagic in der Version 14.0 zum Einsatz, zum Testen des Verhaltensmodells und zur Erzeugung der Testfälle wurden Teile des Modells in IBM/telelogic Rhapsody UML Version 7.1 portiert, da nur diese Software die erforderlichen Funktionen zu Modellausführung, Modelltest und Testfallgenerierung zur Verfügung stellt.

Das Beispielmodell ist dabei so aufgebaut, dass jeder einzelne Prozessschritt archiviert wird, wodurch die Entwicklung des Modells von Anfang bis Ende des Prozesses nachvollzogen werden kann. In den nachfolgenden Unterabschnitten werden einzelne Diagramme des Beispielmodells zur Verdeutlichung einzelner Prozessschritte genutzt.

6.3. Phasen

Der gesamte Erstellungsprozess des Anforderungsmodells ist in fünf wesentliche Phasen gegliedert. Sie sind nach der jeweiligen Kernaufgabe benannt, die in der jeweiligen Phase abgearbeitet wird. Die fünf Phasen sind:

- P0 - Spezifikation des Systemkontextes
- P1 - Spezifikation der Systemfunktionen
- P2 - Spezifikation des Systemverhaltens
- P3 - Spezifikation der Subsystemarchitektur
- P4 - Fortschreibung/Anpasssung des Anforderungsmodells

Die Phasen werden linear nacheinander abgearbeitet. Erst innerhalb der einzelnen Phasen werden die nötigen Iterationsschleifen definiert. Von einer Phase in die nachfolgende Phase darf erst dann gewechselt werden, wenn entsprechende Phasenendkriterien erreicht worden sind.

Die letzte Phase P4 nimmt eine Sonderstellung ein: Sie endet erst dann, wenn feststeht, dass am Anforderungsmodell keinerlei Änderungen mehr vorgenommen werden. Diese Situation ergibt sich gemäß Abschnitt 6.1.3 jedoch erst nach Ende der Lebensdauer des betrachteten Systems, da auch jede Modifikation während des Betriebs, Änderungen durch neue Randbedingungen oder neu hinzugefügte Funktionen zu einer Änderung des Anforderungsmodells führen müssen. Daher ist die Phase P4 diejenige mit der größten Lebensdauer innerhalb des Prozesses.

In den folgenden Unterkapiteln werden die einzelnen Phasen detailliert beschrieben.

6.3.1. P0 - Spezifikation des Systemkontextes

Ziel der Phase P0 „Spezifikation des Systemkontextes" ist der Einstieg in die Modellierung. Entsprechend den Festlegungen in Abschnitt 6.2.3 erfolgt dieser Einstieg durch die Erstellung einer ersten, groben Abgrenzung des zukünftigen Systems von seiner Umwelt. Innerhalb der Phase 0 wird eine erste Fassung der Systemabgrenzungssicht erstellt. Der einzige durchlaufene Subprozess in dieser Phase ist daher der Subprozess S1 „Systemabgrenzungssicht erstellen/überarbeiten" (für dessen Definition siehe Abschnitt 6.4.1). Sinnvollerweise wird pro Iterationsdurchlauf ein Akteur zur Systemabgrenzungssicht hinzugefügt. Wenn keine weiteren Akteure ermittelt werden können, kann die Iteration und damit die gesamte Phase P0 beendet werden.

Die Prozessdefinition der Phase 0 zeigt die Grafik in Bild 6.9.

Ergebnis dieser ersten Prozessphase ist die Verdeutlichung des Systemkontextes und eine erste Abschätzung, wie viele und welche externen Akteure mit dem zukünftigen System interagieren werden. Daraus lässt sich beispielsweise ableiten, welche Fachabteilungen in den weiteren

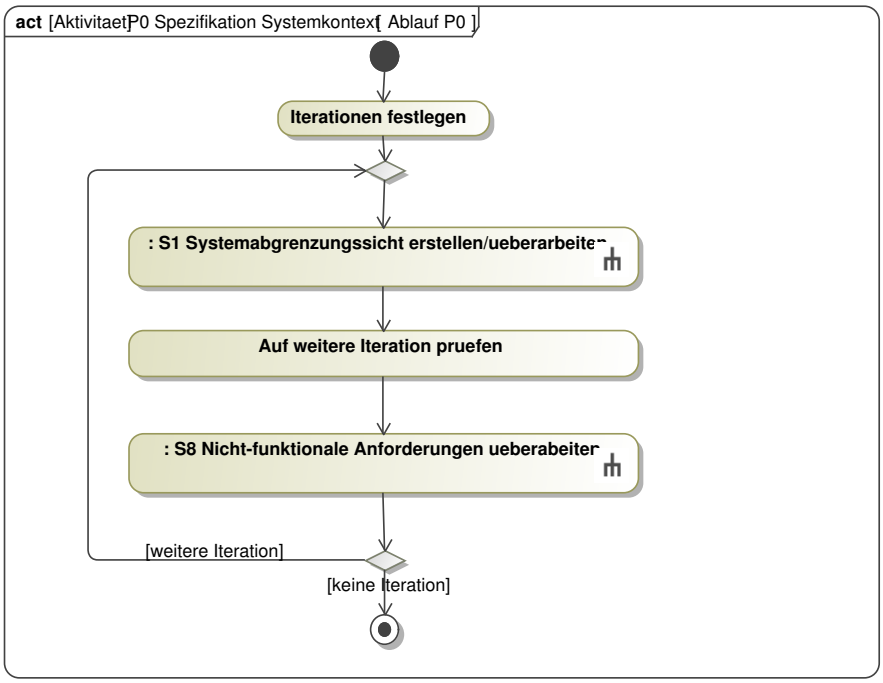


Abbildung 6.9.: Ablauf Phase P0

Prozess der Anforderungserstellung einbezogen werden sollten. Würde beispielsweise bei der ersten groben Systemabgrenzung eines eigentlich nicht sicherheitskritischen Dispositionssystems festgestellt, dass Daten von einem sicherheitsrelevanten System bezogen werden, so könnte aus dieser Information gefolgert werden, dass z.B. Überlegungen zur Rückwirkungsfreiheit mit der Fachabteilung des sicherheitskritischen Systems angestellt werden müssten.

In Tabelle 6.2 sind die wesentlichen Kriterien zur Durchführung der Phase 0 nochmals zusammengefasst.

Aspekt	
Iteration	Iteration erfolgt über die einzelnen Akteure in der Systemumgebung
Iterationsendkriterium	Wenn zunächst keine weiteren Akteure für die Systemumgebung ermittelt werden können, kann die Iterationsschleife abgebrochen werden
Phasenendkriterium	Siehe oben

Tabelle 6.2.: Prozesskriterien Phase 0

6.3.1.1. Beispielmodell

Wie beschrieben, stellt die Phase 0 den Einstieg in die Modellierung dar, weswegen sich die Arbeiten in dieser Phase noch nicht auf bestehende Artefakte stützen können. Man beginnt vielmehr auf einem leeren Blatt Papier bzw. mit einer leeren Projektdatei im Modellierungswerkzeug. Sicherlich werden aber außerhalb des eigentlichen Modellierungsprozesses bereits

Ideenskizzen, implizite Vorstellungen und informelle Anforderungen existieren. Diese sind wesentliches Ausgangsmaterial für die Arbeiten in dieser Phase, für deren erste Schritte sich durchaus auch das Brainstorming in einer Gruppe anbietet.

Im Beispielmodell wird zu Beginn der Phase 0 zunächst das SuB modelliert, wozu ein entsprechend stereotypisierter Block mit Namen „BUe" angelegt wird. Dieser steht für die gesamte zu entwickelnde Bahnübergangssicherungsanlage, inklusive aller physischen Komponenten, wie Signalgebern, Schrankenanlage, Überwachungssignalen, etc. In insgesamt sechs Iterationen wurden durch Ausführung der Aktivitäten des Subprozesses S1 daraufhin die das System umgebenden Akteure eingeführt und die jeweiligen Kommunikationsbeziehungen mit dem BÜ festgelegt. Dazu werden für jeden Akteur zunächst die Schnittstellen zum BÜ ermittelt und als Ports in das Modell aufgenommen. So wird beispielsweise - entsprechend der Systembeschreibung in Abschnitt 6.2.6 - der Triebfahrzeugführer (Tf) eines sich nähernden Schienenfahrzeuges durch Überwachungssignale über den Sicherungszustand des Bahnübergangs informiert. Jedes Überwachungssignal stellt im funktionalen Modell somit eine Schnittstelle zum Akteur „Triebfahrzeugführer" dar, und wird daher über einen Port an der Systemgrenze modelliert (Port „UeS"). Entsprechend der genannten Randbedingungen kann der BÜ je nach Gleistopologie über zwei bis vier Überwachungssignale verfügen, weswegen der Port „UeS" die Multiplizitätsangabe „2...4" trägt. Die Kommunikationsflüsse zwischen Akteuren und BÜ werden in dieser frühen Phase noch sehr abstrakt formuliert. Für das Überwachungssignal wird beispielsweise lediglich modelliert, dass dieses den generellen Befahrbarkeitszustand des BÜ an den Tf überträgt.

Die zu entwickelnde Bahnübergangssicherungsanlage soll zugbewirkt ein- und ausgeschaltet werden. Als weitere Schnittstellen des Systems nach aussen werden daher Sensoren benötigt, die erkennen, dass sich ein Schienenfahrzeug nähert bzw. dass dieses den Bahnübergang komplett geräumt hat. Diese Schnittstellen, „Einschaltsensor" und „Ausschaltsensor", werden vom Akteur „Schienenfahrzeug" beeinflusst, da dessen Anwesenheit am jeweiligen Sensor vom System als „Annäherungsinformation" bzw. der „Räumungsinformation" interpretiert wird. Auch hier werden die Schnittstellen im Modell durch Ports repräsentiert, die die entsprechenden Multiplizitäten gemäß der Systemskizze in Abbildung 6.8 besitzen und die oben genannten. Informationen entgegen nehmen.

Weiterhin lassen sich die Straßenverkehrsteilnehmer als weitere wichtige Akteure im Systemumfeld identifizieren. Diese werden - analog zum Triebfahrzeugführer - durch den BÜ über den aktuellen Befahrbarkeitszustand der Straßenquerung informiert. Dies geschieht durch die Lichtzeichen- und Schrankenanlage, die durch entsprechende Ports („LZA" und „Schrankenanlage") an der Systemgrenze dargestellt werden müssen. Beide unterscheiden sich zwar deutlich in ihrer physikalischen Wirkungsweise, dennoch übermitteln beide eine ähnliche Information vom System an den Verkehrsteilnehmer. Dies wird im Modell durch den gleichen Informationsfluss „Befahrbarkeit Straße" ausgedrückt.

Eine Sonderstellung nimmt der Akteur „Umwelt" ein, der die sonstige Umgebung des Systems und insbesondere die einwirkenden Umweltbedingungen wie Temperatur, Feuchtigkeit, Erschütterungen und elektromagnetische Abstrahlung darstellt. Diese Umweltbedingungen wirken allerdings nicht über eine abgrenzbare Schnittstelle auf das System ein, sondern diffus auf sämtliche Komponenten. Daher wird für die „Umweltbedingungen" im funktionalen Modell kein Port vorgesehen. Diese interagieren vielmehr direkt mit dem Block „BUe", was andeutet, dass das gesamte beschriebene System betroffen ist. Über den Subprozess S8 (siehe Abschnitt 6.4.8) werden anschließend nicht-funktionale Anforderungen - wie beispielsweise Angaben zu ertragbaren Temperaturbereichen - entsprechend dem in 5.2 beschriebenen Konzept mit dem Informationsfluss vom Umwelt-Akteur zum SuB verknüpft. Dadurch wird im Modell ausgedrückt, dass die Umweltbedingungen durch die entsprechenden nicht-funktionalen Anforderungen näher spezi-

fiziert werden.

Der Entwicklungsstand des Beispielmodells nach Abschluss der Phase P0 kann dem Bild 6.10 entnommen werden. Der Arbeitsstand erhebt übrigens keinerlei Anspruch auf Vollständigkeit bezüglich Systemgrenzen, Ports und Akteuren. Er ist vielmehr als Grundlage zu verstehen, die in den folgenden Phasen weiter verfeinert und detailliert wird. Genau dieses allmähliche Anreichern des Modells ist die wesentliche Konzeptidee hinter dem verwendeten iterativ-inkrementellen Entwicklungsprozess.

6.3.2. P1 - Spezifikation der Systemfunktionen

Im Fokus der Phase P1 steht die Ableitung der Systemfunktionen, wobei die in der vorherigen Phase P0 erstellte Systemabgrenzung als Bezugsbasis verwendet wird. Ziel der Phase 1 ist die Modellierung der Systemfunktionen in Form von Anwendungsfällen und damit die initiale Erstellung der Systemfunktionssicht gemäß Abschnitt 5.1.4.2. Die Phase P1 umfasst die Ausführung der Subprozesse S1, S2 und S8. Im Subprozess S2 wird dabei die in dieser Phase neu hinzukommende Systemfunktionssicht iterationsweise aufgebaut. In den Subprozessen S1 und S8 werden die Systemabgrenzungssicht und die Anforderungsschnittstellensicht entsprechend den Konsistenzbedingungen nachgezogen.

Der sich daraus ergebende Ablauf der Phase P1 ist in Bild 6.11 dargestellt.

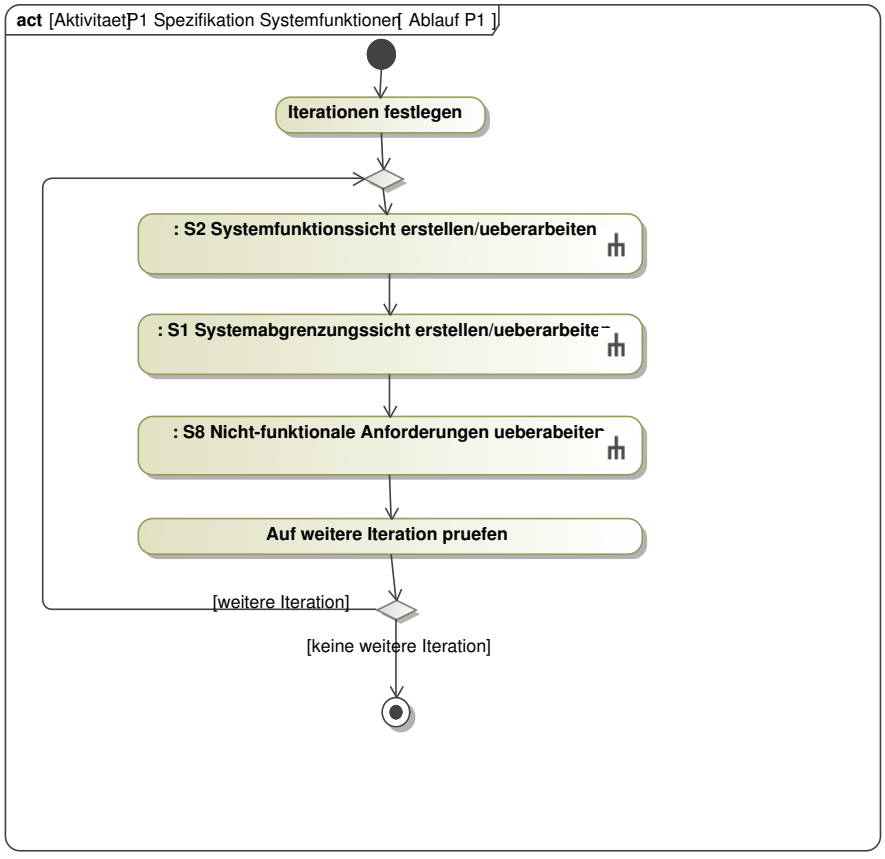


Abbildung 6.11.: Ablauf Phase P1

Zu Beginn der Phase wird zunächst in der Aktion „Iteration festlegen” das Iterationskonzept für

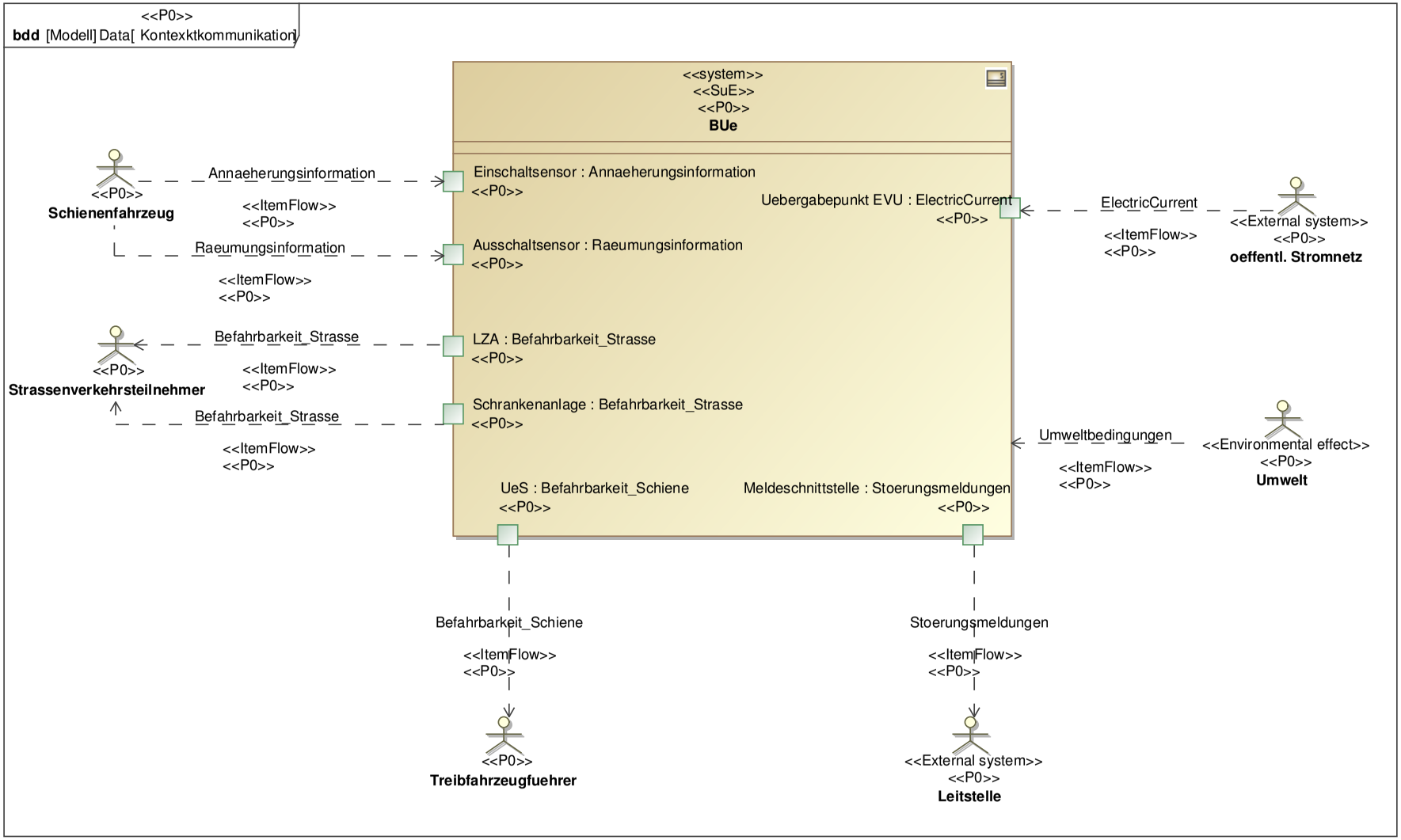


Abbildung 6.10.: Systemabgrenzungssicht nach Abschluss der Phase 0

den weiteren Ablauf definiert. Das Iterationskonzept ist nicht von vornherein explizit festgelegt, vielmehr muss je nach konkretem Projekt über ein passendes Vorgehen entschieden werden. Im Folgenden wird jedoch ein Verfahren vorgestellt, dass sich bei der Ableitung von Systemfunktionen als praktikabel herausgestellt hat. Es basiert auf der schrittweisen Verfeinerung zunächst sehr grober Anwendungsfälle durch Aktivitätsdiagramme.

Ausgangspunkt ist der denkbar abstrakteste Anwendungsfall, der alle Funktionen repräsentiert, die das System während seines Lebenszyklus ausführen wird: der Anwendungsfall „Lebenszyklusfunktionen ausführen". Zu diesem Anwendungsfall wird im ersten Schritt ein Aktivitätsdiagramm entworfen, das die auf oberster Ebene differenzierbaren Aktionen des Systems umfasst. Üblicherweise sind dies Aktivitäten zur Inbetriebnahme, zur Abwicklung von Regelbetrieb und zum Verhalten in Störungsfällen sowie zur Ausserbetriebnahme des Systems. Jede dieser Aktivitäten wird anschließend in einen neuen Anwendungsfall überführt, der über eine dependency-Relation mit dem übergeordneten Anwendungsfall verknüpft wird. In der nächsten Iteration werden dann für alle soeben abgeleiteten Anwendungsfälle wiederum Aktivitätsdiagramme erstellt, durch die eine weitere Detaillierung der Anwendungsfälle erzielt wird.

Bild 6.12 zeigt dieses Vorgehen. Für den globalen Anwendungsfall wird ein Aktivitätsdiagramm erstellt. Die einzelnen Aktivitäten innerhalb dieses Diagramms bilden dann die Anwendungsfälle der folgenden Stufe. Dieses Vorgehen wird so lange wiederholt, bis sich hinreichend feine Anwendungsfälle herausgebildet haben. Es ergibt sich somit ein hierarchischer Baum von zunehmend detaillierten Anwendungsfällen, wobei die Blätter des Baumes durch die detailliertesten Anwendungsfälle gebildet werden. Zusätzlich erzeugt dieses Verfahren einen Anwendungsfall-bezogenen Baum von Aktivitätsdiagrammen, der als Ausgangspunkt für die detaillierte Verhaltensspezifikation in Phase P2 dienen kann (siehe Abschnitt 6.3.3 und 5.1.4.4.5).

Dieses Konzept der rekursiven Ableitung der Anwendungsfälle muss für die Abbildung im Prozessmodell nun in einzelne, nacheinander ausführbare Iterationen transformiert werden. Das geschieht durch eine entsprechende Verwendung des Subprozesses „S2 - Systemfunktionssicht erstellen/überarbeiten": Bei jeder Ausführung dieses Subprozesses wird ein Anwendungsfall aus einem Aktivitätsdiagramm eines bereits bestehenden Anwendungsfalls neu erstellt und im selben Schritt dessen eigenes Aktivitätsdiagramm modelliert. Dadurch läst sich die Verfeinerung der Anwendungsfälle bis zum Erreichen der gewünschten Granularität prinzipiell beliebig fortsetzen. Weitere Details zum Subprozess S2 finden sich in Abschnitt 6.4.2.

Die Tabelle 6.3 fasst zusammen, welche Kriterien für die Iterationen und Abbruchkriterien der Phase 1 gelten.

Aspekt	
Iteration	Die Iteration erfolgt über die einzelnen Systemfunktionen. In jeder Iteration wird dabei eine neue Systemfunktion abgeleitet und diese gleichzeitig durch Aktivitätsdiagramme genauer spezifiziert.
Iterationsendkriterium	Ausreichende Granularität der Menge aller Systemfunktionen.
Phasenendkriterium	Siehe Iterationsendkriterium

Tabelle 6.3.: Prozesskriterien Phase 1

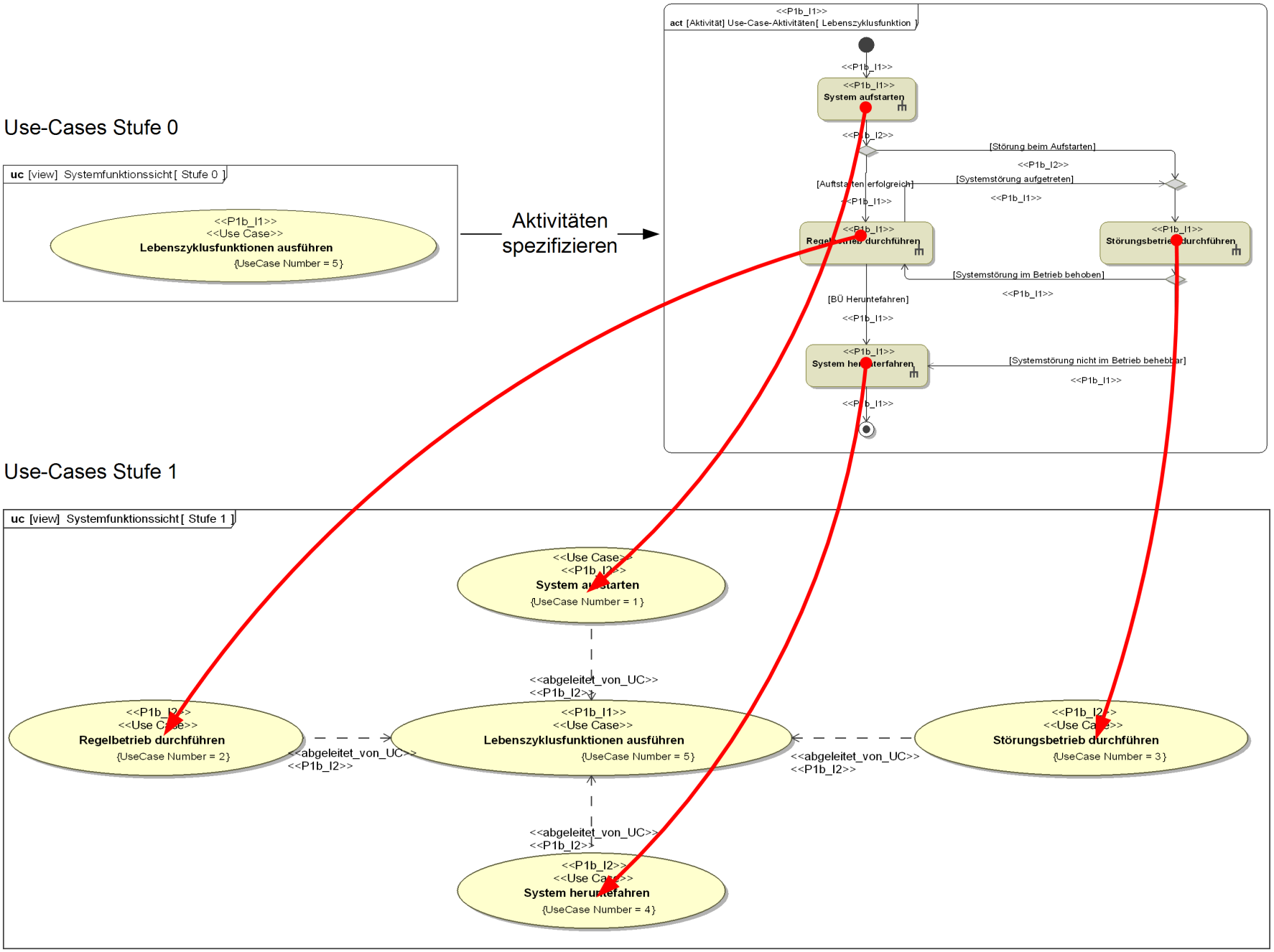


Abbildung 6.12.: Detaillierung von Anwendungsfällen über Aktivitätsdiagramme

6.3.2.1. Beispielmodell

Auch im Beispielmodell werden die Anwendungsfälle nach dem zuvor beschriebenen Schema abgeleitet. Ausgangspunkt für den dabei entstehenden Baum der Anwendungsfälle ist dabei die Frage, welche Funktionen die Bahnübergangssicherungsanlage ihrer Umwelt während ihres Lebenszyklus anbieten soll. Die Wurzel des Baumes bildet dabei auf der in 6.3.2 angesprochene, virtuelle Anwendungsfall „Lebenszyklusfunktionen ausführen", der stellvertretend für alle Lebenszyklus-Funktionen steht.

Für die Bahnübergangssteuerung lassen sich als nächste Verfeinerungsebene vier Anwendungsfälle ableiten, die wesentliche Eckpunkte des Systemlebenszyklus darstellen. Eine Übersicht über diese Anwendungsfälle als Use-Case-Diagramm findet sich in Bild 6.12. Die Eigenschaften dieser Anwendungsfälle lassen sich wie folgt beschreiben:

- Es existiert ein Anwendungsfall, der das Aufstarten des Systems beschreibt. Dieser Anwendungsfall kapselt alle Funktionen, die nötig sind, um ein ausgeschaltetes System in einen betriebsfähigen Zustand zu überführen. Dazu gehören beispielsweise das Durchführen von Selbsttests, das Einlesen von Konfigurationsparametern und das Herstellen eines definierten Ausgangszustandes. Akteure, die mit diesem Use-Case verbunden sind, sind beispielsweise Instandhaltungspersonale, die das Aufstarten des Systems anfordern und begleiten. Im Fall des BÜ sind dies z.B. Mitarbeiter der LST-Instandsetzung, die eine neue oder nach Umbauten modifizierte Bahnübergangssicherungsanlage konfigurieren und wieder in Betrieb nehmen. Ebenfalls Bestandteil dieses Anwendungsfalls ist das Herstellen der Grundstellung des Bahnübergangs (LzA dunkel, Schranken in oberer Endlage, Überwachungssignal dunkel). Im Beispielmodell trägt dieser Anwendungsfall den Namen „System aufstarten".
- Ein ähnlicher Anwendungsfall beschreibt das geordnete Herunterfahren der Bahnübergangssicherungsanlage, z.B. zu Wartungszwecken oder bei einer geplanten Ausserbetriebnahme. Auch hier kann der assoziierte Akteur beispielsweise ein Wartungstechniker sein, der das geordnete Abschalten des Systems durchführt. Oftmals wird - ebenso wie beim vorhergehenden Anwendungsfall - auch hier zuvor ein bestimmter Anlagenstatus herbeigeführt, aus dem heraus das System sicher abgeschaltet werden kann. Im Beispielmodell übernimmt der Anwendungsfall „System herunterfahren" diese Rolle und kapselt die Funktionalität, die für ein geregeltes Abschalten der BÜ-Sicherungsanlage vom System bereitgestellt wird.
- Die eigentlichen produktiven Systemfunktionen werden durch den Anwendungsfall „Regelbetrieb durchführen" abgebildet. Dieser repräsentiert stellvertretend alle Aktivitäten, die als Dienstleistungen der Umwelt angeboten werden, so lange sich das System selbst in einem voll funktionsfähigen, ordnungsgemäßen Zustand befindet. Im Fall der Bahnübergangsicherungsanlage sind dies die folgenden Funktionen:
  - Sichern des Bahnübergangs durch die zugbewirkte Einschaltung
  - Sichern des Bahnübergangs durch die HET
  - Wiederherstellen der Grundstellung durch das Freifahren des BÜ
  - Wiederherstellen der Grundstellung durch das Ablaufen der Grundstellungszeit

  Mit diesem Anwendungsfall sind üblicherweise alle Akteure verknüpft, die von der angebotenen Dienstleistung Gebrauch machen. Im Fall des BÜ sind dies die Straßenverkehrsteilnehmer (fordern vom BÜ die Anzeige des Befahrbarkeitszustandes), das Schienenfahrzeug (schaltet die Sicherungsanlage ein und aus) und der Triebfahrzeugführer (fordert

vom BÜ die Anzeige des Befahrbarkeitszustandes und fordert via HET gegebenenfalls die Sicherung an).

- Sollte system-intern ein Fehler auftreten, so muss das System auf diesen Fehler in geeigneter Weise reagieren und beispielsweise in eine sichere Rückfallebene fallen. Im Fall der Bahnübergangssicherungsanlage muss unter anderem beim Ausfall von einer entsprechenden Anzahl an Rotlichtern auf einer Seite zwar der noch anstehende Sicherungszyklus beendet werden, dann aber der Bahnübergang für weitere Sicherungen gesperrt werden, da der Sicherungszustand für den Kraftfahrzeugverkehr nicht mehr eindeutig erkennbar ist. Diese Reaktionsmöglichkeit auf Fehler wird im Baum der Anwendungsfälle durch den Use-Case „Stoerungsbetrieb durchführen" dargestellt. Dieser kapselt die einzelnen Funktionen, die für die Reaktion auf Systemfehler erforderlich sind.

Nach Ableitung dieser vier grundlegenden Anwendungsfälle wurden diese in weiteren Iterationen verfeinert. Für den Anwendungsfall „Regelbetrieb durchführen" wurden so z.B. die Anwendungsfälle „Annäherung Sfz überwachen", „Räumung BUe ueberwachen", „BUe sichern", „Grundstellung herstellen" und „Selbsttest durchfuehren" abgeleitet. Diese lassen sich auf Ebene der Anwendungsfälle nicht mehr sinnvoll verfeinern und bilden damit die Blätter des Anwendungsfall-Baumes. Sie stellen somit die atomaren Funktionen dar, die die Bahnübergangssicherungsanlage ihrer Umgebung anbietet.

Insgesamt ergeben sich für alle Zweige des Baumes nach Abschluss des Prozessschrittes P1 insgesamt 13 Anwendungsfälle. Davon bilden 10 Anwendungsfälle die atomaren Systemfunktionen. Dies ist in Bild 6.13 dargestellt. Während des Prozessablaufes wurde zudem festgestellt, dass für die ordnungsgemäße Funktion ein weiterer Akteur („LST-Mitarbeiter") für Wartungsarbeiten, zur Störungsbehebung und Konfiguration des Systems erforderlich ist. Der neue Akteur wurde daraufhin in der Systemfunktionssicht eingeführt. Gemäß dem Konsistenzkriterium für Akteure (siehe Abschnitt 5.1.5.2) muss er auch in der Systemabgrenzungssicht enthalten sein. Bei der nächsten Ausführung des Subprozesses S1 wurde er dann auch in die Systemabgrenzungssicht aufgenommen, wo er in Bild 6.20 zusammen mit den für die Kommunikation erforderlichen Schnittstellen zu erkennen ist. Diese Darstellung zeigt anschaulich, wie sich bereits bestehende Modellartefakte durch das weitere Fortschreiten des Prozesses ändern und an vorgenommene Modifikationen und Ergänzungen anpassen.

6.3.3. P2 - Spezifikation des Systemverhaltens

Ziel der Phase P2 ist ein ausführbares und funktionsgetestetes Anforderungsmodell. Am Ende der Phase soll das Verhalten aller Systemfunktionen ausmodelliert und das Modell ausführbar sein. Integriert in diese Phase ist zudem eine Prüfung des Modellverhaltens durch Testfälle und - wenn technisch verfügbar - auch durch eine formale Verifikation gegen Sicherheitsbedingungen. Nebenziel der Phase 2 ist die Schaffung einer möglichst breit abdeckenden Basis von Systemtestfällen.

Zur Erreichung des Ziels dieser Phase werden im Modell enthaltene Systemfunktionen zunächst durch Interaktionsszenarien genauer spezifiziert, wodurch die Szenariensicht (siehe 5.1.4.3) aufgebaut wird. Parallel dazu wird das Systemverhalten spezifiziert, so dass ein ausführbares Anforderungsmodell entsteht. Aus diesem iterativ entstehenden Verhaltensmodell werden gleichzeitig Testfälle generiert, die in der jeweils nachfolgenden Modellierungsiteration für Regressionstests verwendet werden können. Nach Abschluss der Modellierung stellen sie die Testbasis für Systemtestfälle dar.

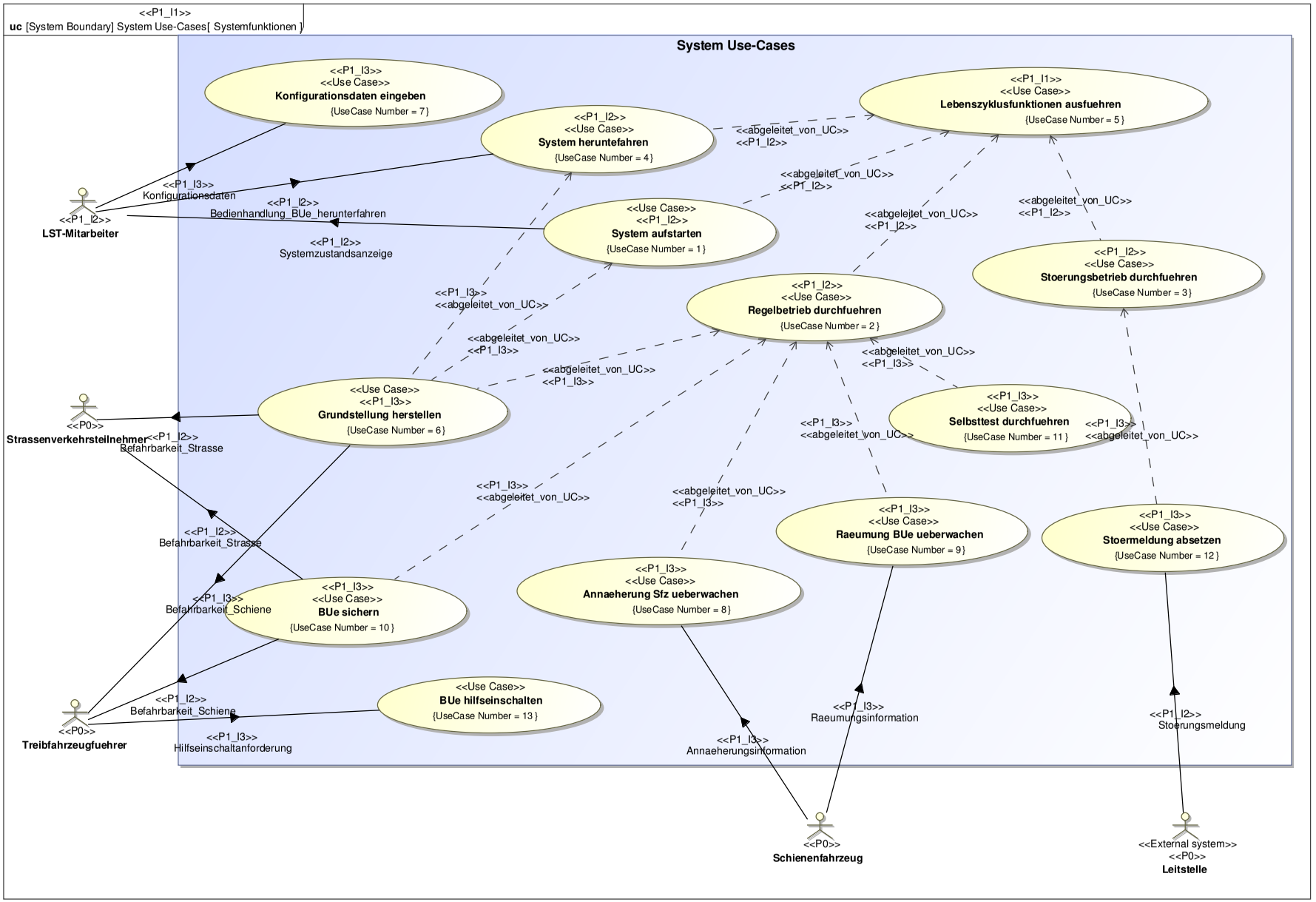


Abbildung 6.13.: Systemfunktionssicht nach Ende der Phase 1

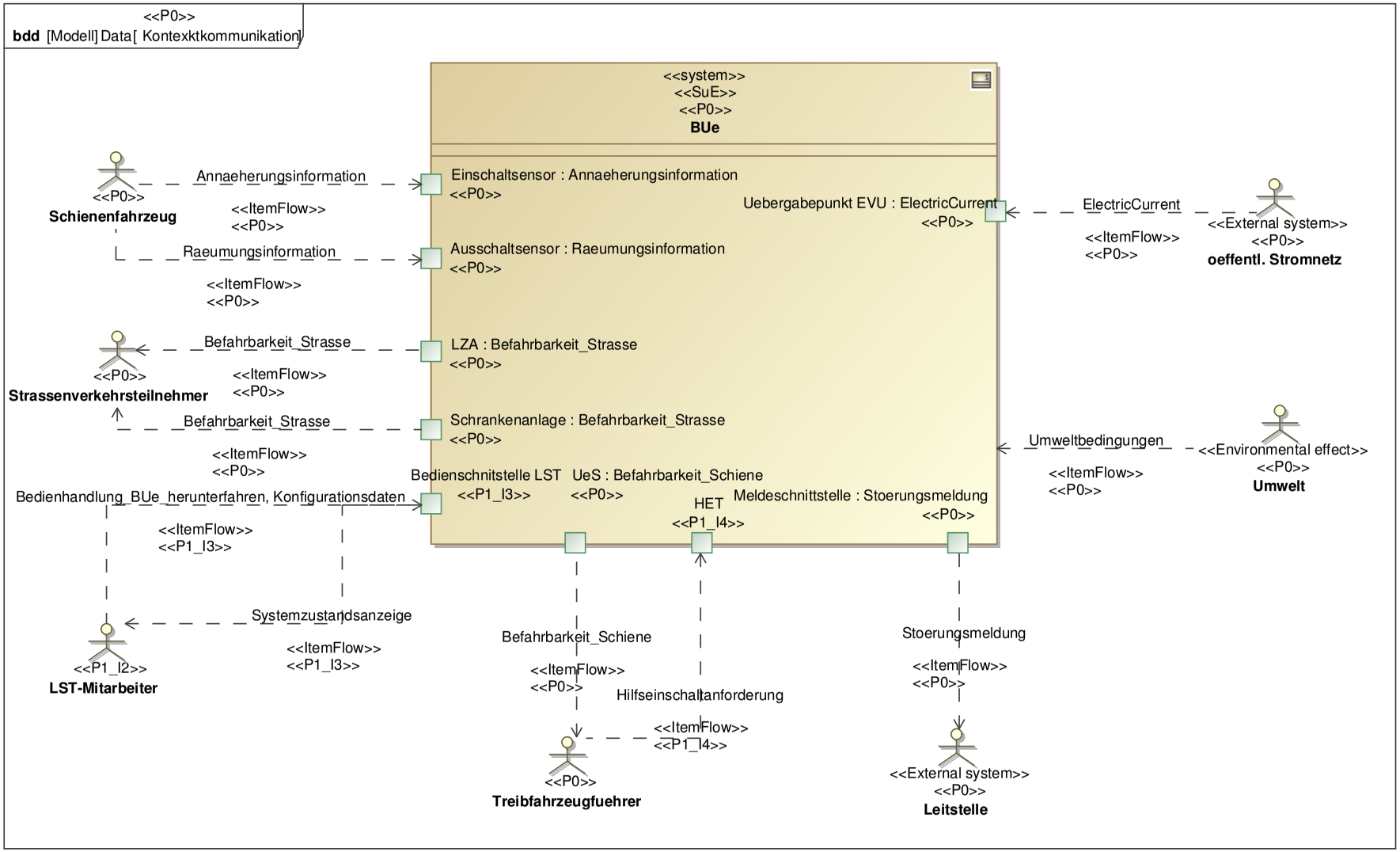


Abbildung 6.14.: Systemabgrenzungssicht nach Ende Phase 1 mit Modifikationen gegenüber Phase 0

Die drei wesentlichen Tätigkeiten innerhalb der Phase werden in den folgenden Unterabschnitten näher beschrieben.

6.3.3.1. Prinzipielles Vorgehen

Die Spezifikation des Systemverhaltens erfolgt iterativ in zwei ineinander geschachtelten Schleifen. Die äußere Schleife läuft über alle Systemfunktionen aus der vorhergehenden Phase P1, wobei pro Schleifendurchlauf die Szenarien und das gewünschte Verhalten der Funktion modelliert werden.

Die einzelnen Interaktionsszenarien für jede Systemfunktion und die dazu nötigen Verhaltensmodellelemente werden dann innerhalb der inneren Schleife erstellt. Jeder neue Schleifendurchlauf ergänzt dabei ein Szenario durch Ausführung des Subprozess „S3 - Szenariensicht erstellen/überarbeiten" gemäß dessen Definition in Abschnitt 6.4.3. Im Subprozess „S4 - Systemverhaltenssicht erstellen/überarbeiten" wird anschließend das Systemverhalten mit Aktivitätsdiagrammen oder Zustandsautomaten so modelliert, dass alle zu diesem Zeitpunkt existierenden Interaktionsszenarien erfüllt werden. Die Überprüfung, ob Interaktion und modelliertes Verhalten tatsächlich übereinstimmen, erfolgt im anschließenden Subprozess „S5 - Systemverhalten testen", in welchem die Interaktionsszenarien als Testfälle für das Systemverhalten herangezogen werden. Wird es für das Modell als erforderlich angesehen, kann zusätzlich auch eine formale Verifikation des Modells gemäß Subprozess „S6 - Systemverhalten verifizieren" durchgeführt werden. Diesen prinzipiellen Ablauf zeigt das Aktivitätsdiagramm der Phase in Bild 6.15.

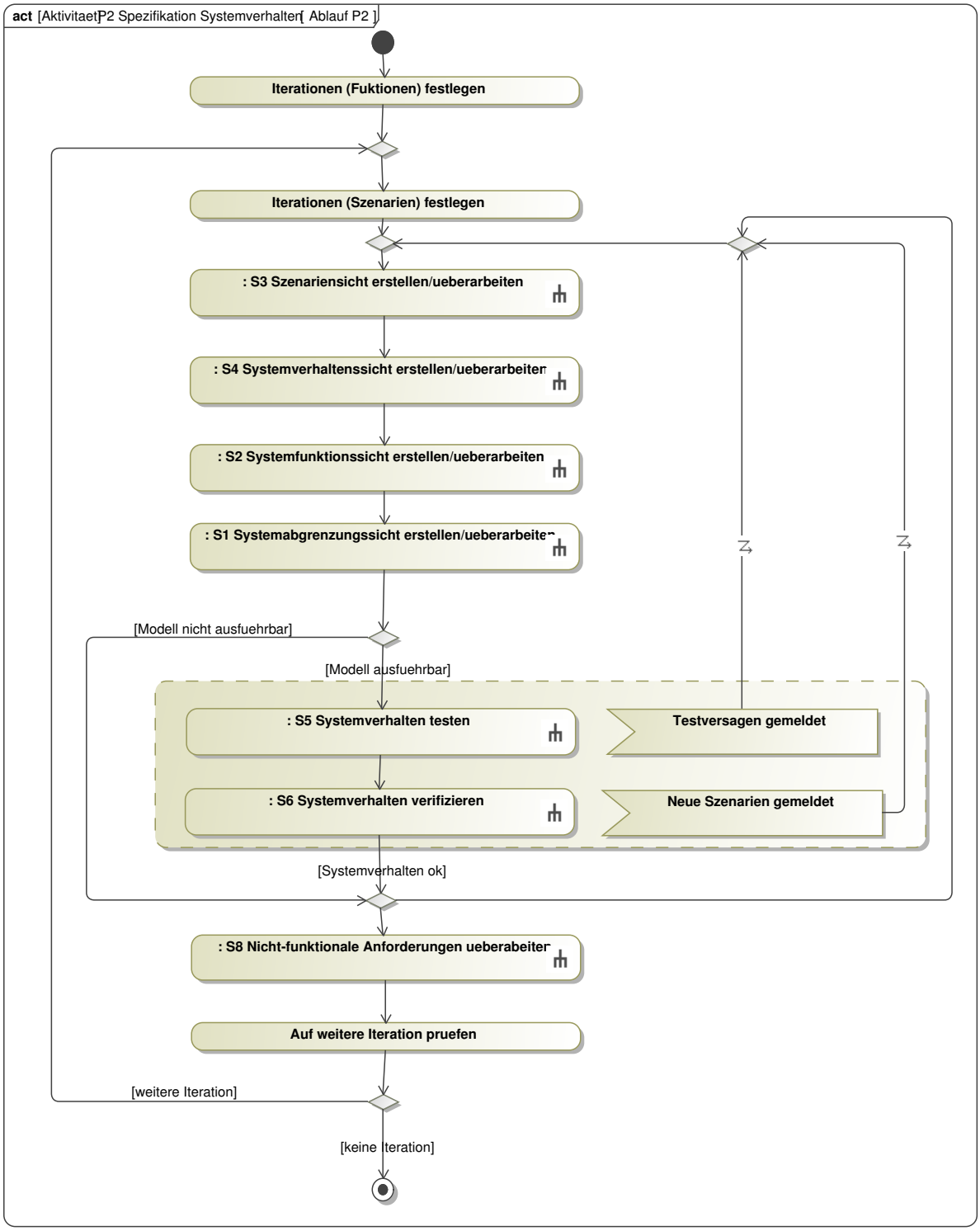


Abbildung 6.15.: Ablauf Phase P2

In jedem Durchlauf der äußeren Schleife erfolgt somit eine schnelle Abfolge von Spezifikation des Soll-Verhaltens, Implementierung dieses Verhaltens und Test des Verhaltens gegen die Szenarien. Die Abläufe in dieser Phase entsprechen somit einem minimalistischen V-Modell, wobei der linke Schenkel des V die Verhaltensimplementierung bildet, die dann im rechten Schenkel

gegen die Szenarien validiert wird. Dies entspricht der zitierten Aussage von Cockburn, dass jeder inkrementelle Prozess einem mehrfachen Durchlauf kleiner V-Prozesse entspricht. Ein solches minimalistisches V-Modell ist in Bild 6.16 dargestellt.

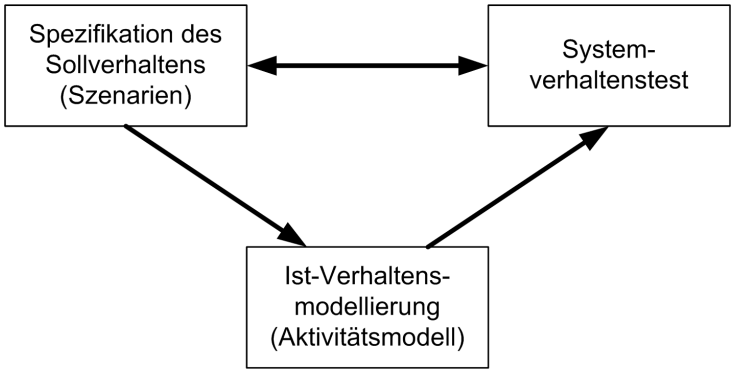


Abbildung 6.16.: V-Modell der Verhaltensmodellierung

Mit jedem Schleifendurchlauf nehmen damit der Funktionsumfang des Modells und die Anzahl der Szenarien zu, bis letztendlich alle Systemfunktionen sowohl durch Interaktionszenarien spezifiziert sind, als auch in Verhaltenskonstrukte umgesetzt wurden. Durch die Wahl kleiner Inkremente zwischen den einzelnen Schleifendurchläufen, die Nutzung von Testausführungswerkzeugen und vor allem durch die Anwendung automatischer Testfallgenerierungstechniken wird dabei zum einen die Konsistenz zwischen Szenariensicht und Systemverhaltenssicht sichergestellt, zum anderen ergibt sich eine zusätzliche Kontrollmöglichkeit für die händisch erstellten Szenarien. Dazu ist ein Mechanismus vorgesehen, der beim Auftreten bestimmter Ereignisse die aktuelle Iteration unterbricht und eine Korrektur des Verhaltensmodells und eine Ergänzung der Szenarien ermöglicht.

6.3.3.2. Ableitung der Szenarien

In der inneren Iterationsschleife werden die Szenarien für die jeweils aktuelle Systemfunktion erstellt. Dabei müssen für jede Systemfunktion sowohl der reguläre, ungestörte Ablauf, wie auch alle relevanten Störungsfälle beschrieben werden. Während die Modellierung des Regelablaufs üblicherweise relativ leicht fällt, besteht die Schwierigkeit darin, möglichst viele denkbare Störungsfälle zu ermitteln. Ein Ansatz dafür kann in der schematisierten Analyse des jeweiligen Regelablaufs und einer Variation einzelner Nachrichten bestehen. In einem solchen Ansatz könnte beispielsweise für jede Nachricht im Regelablauf geprüft werden, welche Auswirkungen sich ergeben, wenn:

- die Nachricht nicht gesendet wird
- die Nachricht zu spät gesendet wird
- die Nachricht zu früh gesendet wird
- die Attribute der Nachricht verfälscht sind

Anhand der Formulierung des in der jeweiligen Situation korrekten Systemverhaltens lässt sich vom Regelablauf eine Reihe von Störungsabläufen ableiten. Die exakte Ausgestaltung dieses Prozederes soll jedoch im Rahmen dieser Arbeit nicht weiter beschrieben werden. Vielmehr bleibt an dieser Stelle Raum für weitere Forschungs- und Entwicklungsarbeit. Langfristig sollte als

Entwicklungsziel angestrebt werden, die Menge der händisch erstellten Szenarien möglichst zu begrenzen, da deren Erstellung zeitaufwändig und fehleranfällig ist. Vielversprechend erscheint hingegen der Ansatz, eine kleine Menge händisch erstellter Szenarien durch den wechselweisen Prozess von Verhaltensmodellierung und Testfallgenerierung allmählich zu vervollständigen. Dieses Vorgehen wird in Kapitel 6.4.5 im Rahmen des Test-Subprozesses beschrieben.

Die eigentliche Erstellung der Szenarien erfolgt durch den Subprozess S3, die Umsetzung in Systemverhalten im Subprozess S4. Durch die anschließende Ausführung der Subprozesse S2, S1 und S8 wird die Konsistenz aller anderen Sichten zu dem momentanen Entwicklungsstand der Szenariensicht sichergestellt.

6.3.3.3. Verhaltensmodellierung

Die Verhaltensmodellierung erfolgt durch die Ausführung des Subprozesses „S4 - Systemverhaltenssicht erstellen/überarbeiten" (siehe Abschnitt 6.4.4), wobei die Modellierung des Verhaltens - aufgrund der Ankopplung an die äußere Iterationsschleife - analog zur Struktur des Systemfunktionsbaumes aus der Phase P1 (siehe Abschnitt 6.3.2) erfolgt. Damit weist das Verhaltensmodell eine zum Funktionsbaum ähnliche, hierarchische Struktur von Aktivitäts- oder Zustandsdiagrammen auf. Allerdings kann das Verhaltensmodell über die Funktionsebene hinaus noch weiter verfeinert werden. Dieser Prozessschritt ergänzt damit den hierarchischen Baum von Aktivitätsdiagrammen, der mit der Modellierung der Systemfunktionen in Phase P1 begonnen wurde (siehe Abschnitt 6.3.2), um weitere tiefere Hierarchieebenen.

Dieser hierarchische Baum lässt sich in der SysML durch ein Dekompositions-Strukturdiagramm darstellen. In diesem werden die einzelnen Aktivitäten durch Blöcke repräsentiert und deren hierarchische Verknüpfungen durch Kompositionen dargestellt. Bild 6.17 zeigt eine Prinzipdarstellung eines solchen Strukturbaums für alle Aktivitäten des Beispielmodells.

Im Beispielmodell wurde für alle 10 atomaren Systemfunktionen das Verhalten ausspezifiziert. Um die Ausführbarkeit des Modells sicherzustellen, wird im Strukturmodell dem Block „SuB" das Verhalten des Verhaltensbaums zugewiesen.

6.3.3.4. Testen des Systemverhaltens und Erzeugung neuer Testfälle

Durch den Test des Systemverhaltens wird überprüft, ob das modellierte Verhalten die in den Interaktionsszenarien spezifizierten Abläufe erfüllt[1]. Damit wird die Konsistenz zwischen Szenariensicht und Systemverhaltenssicht sichergestellt. Weiterhin werden durch Analyse des ausführbaren Modells automatisch Testfälle abgleitet, wobei diese Testfälle genau wie die händisch erstellten Interaktionen durch Sequenzdiagramme repräsentiert werden. Die Gesamtmenge aller Szenarien ergibt sich aus der Vereinigung der manuell erstellten Szenarien mit den durch die Testfallgenerierung automatisch erzeugten Sequenzdiagrammen. Der Test des Systemverhaltens im Subprozesses „S5 - Systemverhalten testen" (siehe Abschnitt 6.4.5), bezieht pro Iteration die folgende Menge an Modellartefakten in den Test ein:

- Die kumulierte Menge der Verhaltensartefakte, also die Gesamtmenge des bis dahin in allen Iterationen implementierten Verhaltens
- Die kumulierte Menge der Interaktionsszenarien, also die Gesamtmenge der in allen Iterationen erstellten Interaktionsszenarien, sowohl händisch, als auch durch Testfallgenerierung.

[1] Dabei können die in den Szenarien auf das SuB zulaufenden Signale als Stimuli und die aus dem SuB herausströmenden Signale als Antworten des Systems verstanden werden.

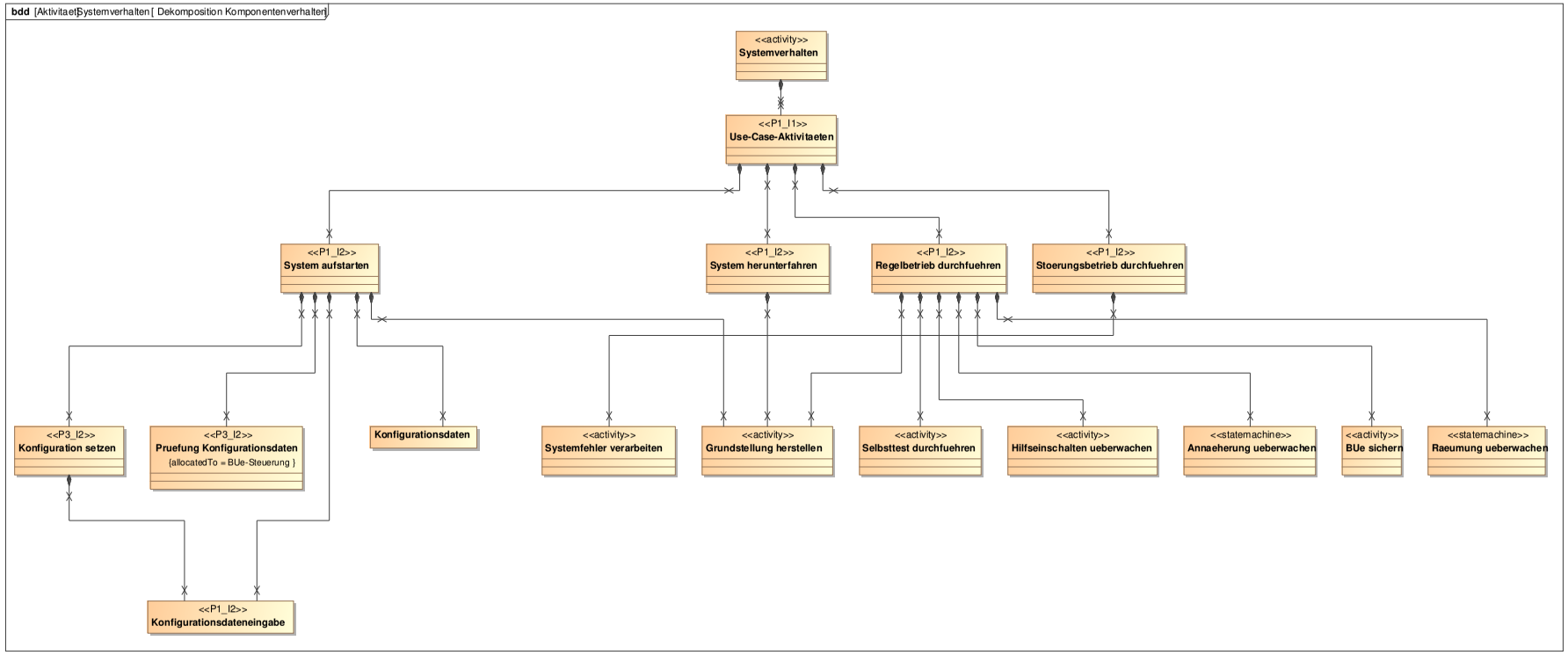


Abbildung 6.17.: Aktivitäts-Hierarchie des Beispielmodells

Als Ergebnis liefert der Subprozess S5 drei mögliche Aussagen:

(a) Das modellierte Systemverhalten ist konsistent zu den Interaktionszenarien und die automatisch erstellten Testfälle offenbaren kein fehlerhaftes oder ungewolltes Verhalten

(b) Das modellierte Systemverhalten ist inkonsistent zu den Interaktionsszenarien

(c) Das modellierte Systemverhalten ist konsistent zu den Interaktionsszenarien, aber die automatisch erstellten Testfälle offenbaren ein ungewünschtes oder fehlerhaftes Verhalten oder zeigen neue, noch unberücksichtigte Szenarien.

Im Fall (a) kann die momentane Iteration fortgesetzt werden, in den Fällen (b) und (c) muss der Grund für die Inkonsistenz bzw. das Fehlverhalten des Modells ermittelt werden. Die nächste Iteration kann erst begonnen werden, wenn der Modellierungsfehler bzw. die Unvollständigkeit der Szenarien behoben wurde.

Im nachfolgenden, optionalen Subprozess „S6 - Systemverhalten verifizieren" kann das Systemverhalten zusätzlich durch eine formale Verifikationstechnik gegen eine Menge von formal definierten Randbedingungen überprüft werden. Hierbei gibt es zwei mögliche Ergebnisse: Entweder das System erfüllt die Randbedingungen, oder es erfüllt sie nicht. Im letzten Fall wird ebenfalls die aktuelle Iteration abgebrochen und der Grund für das Nichtbestehen des Verifikationslaufes muss ermittelt werden. Weitere Informationen zur Durchführung der Verifikation finden sich in Abschnitt 6.4.6. Mangels geeigneter Werkzeuge konnte für das Beispielmodell jedoch keine Verifikation durchgeführt werden.

6.3.3.5. Zusammenfassung

In Tabelle 6.4 werden die wesentlichen Kriterien für den Prozessablauf der Phase 2 zusammengefasst, wobei nach innerer und äußerer Iterationsschleife unterschieden wird.

Aspekt	
Iteration (äußere Schleife)	Diese Iteration läuft über die einzelnen Systemfunktionen. Innerhalb jeder Iteration werden die Interaktionsszenarien für die Systemfunktion erstellt und das Verhalten der jeweiligen Systemfunktion implementiert.
Iterationsendkriterium (äußere Schleife)	Für alle Systemfunktionen wurden die Szenarien erstellt
Iteration (innere Schleife)	Diese Iteration läuft über alle Szenarien, die die aktuelle Systemfunktion der äußeren Schleife detaillieren. In jedem Iterationsdurchlauf wird ein Szenario erstellt.
Iterationsendkriterium (innere Schleife)	Für die momentan bearbeitete Systemfunktion lässt sich kein weiteres Szenario mehr finden oder es ist ein Fehler beim Systemtest gefunden worden oder die Szenarien sind unvollständig
Phasenendkriterium	Die äußere Iterationsschleife wurde vollständig durchlaufen

Tabelle 6.4.: Prozesskriterien Phase 2

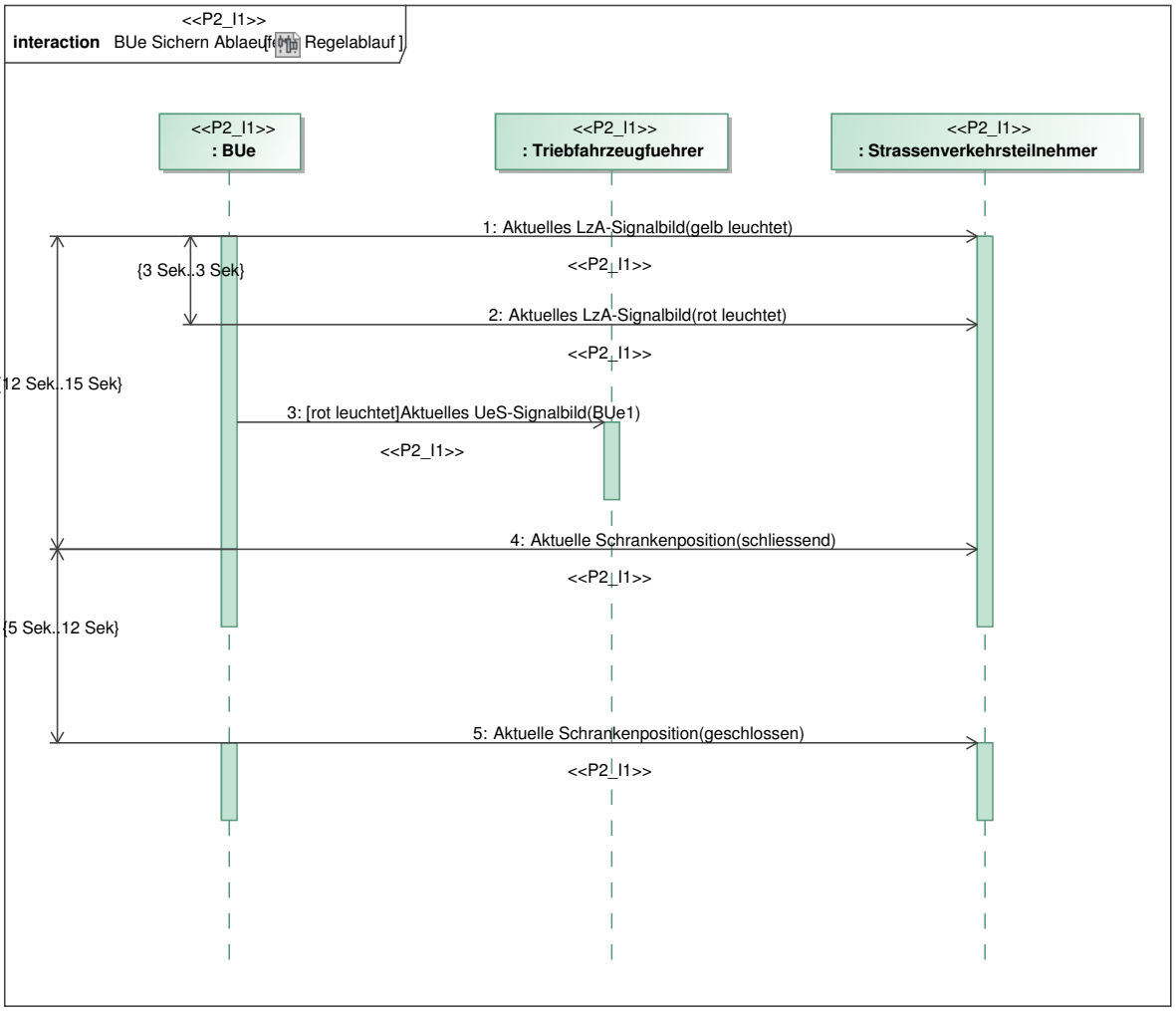


Abbildung 6.18.: Szenariensicht nach Ende der Phase 2

6.3.3.6. Beispielmodell

Im Folgenden wird der gesamte Durchlauf der Phase P2 für das Beispielmodell beschrieben, wobei das in den vorherigen Unterkapiteln vorgestellte, generelle Vorgehen konkret nachvollzogen werden kann.

Um in dieser Phase die Szenarien und die Systemverhaltenskonstrukte zu erstellen, wird zunächst eine Liste aller in der vorherigen Prozessphase P1 identifizierten Systemfunktionen erstellt. Auf jedes Element dieser Liste (also auf jede Systemfunktion) wird daraufhin der nachfolgend beschriebene Prozess angewendet.

Als erster Schritt wird dabei für die jeweils aktuelle Systemfunktion der Regelablauf in einem Sequenzdiagramm modelliert, wodurch sich eine Ablaufbeschreibung entsprechend Bild 6.18 ergibt. Im dargestellten Beispiel der Sicherung des BÜ zeigt das Sequenzdiagramm dabei als Lebenslinien das „SuB" - hier also die Bahnübergangssicherungsanlage - sowie die an der Systemfunktion beteiligten Akteure „Triebfahrzeugführer" und „Strassenverkehrsteilnehmer", die über entsprechende Nachrichten miteinander kommunizieren. Gemäß den Prozessvorgaben muss als nächster Schritt nun ein Verhaltensdiagramm im Modell erstellt werden, dass die im Sequenz-

diagramm dargestellte Soll-Kommunikation hervorruft. Dies geschieht im Beispielmodell durch ein Aktivitätsdiagramm, das zunächst nur den Regelablauf des BÜ-Sicherns implementiert (siehe Bild 6.19).

Wichtig ist bei der Erstellung dieses Diagramms, dass die dort versendeten und empfangenen Signale und deren Attribute - entsprechend der in Kapitel 5.1.5.1 beschriebenen Konsistenzbedingung - exakt so bezeichnet werden, wie in den Nachrichten der Sequenzdiagramme. Nur so kann durch Testausführungswerkzeuge später überprüft werden, ob die durch das modellierte Verhalten hervorgerufene Kommunikation zwischen SuB und Umgebung den Sollvorgaben entspricht. Diese Bedingung wird jedoch durch den datenbankähnlichen Ansatz der meisten Modellierungswerkzeuge ohnehin erzwungen.

Nach Erstellung des Aktivitätsdiagramms wird das Modell ausgeführt[2], das entstehende Kommunikationsverhalten in einer Testumgebung (wie z.B. dem TestConductor für Rhapsody) aufgezeichnet und mit dem zuvor in Form der Sequenzdiagramme modellierten Soll-Kommunikationsverhalten verglichen. Sind sie inhaltlich identisch, erfolgt im nächsten Schritt die Anwendung des automatischen Testfallgenerierungswerkzeuges auf das Verhaltensmodell. Diese hat zum einen das Ziel, implizit mitmodelliertes, gewünschtes Verhalten explizit zu machen, andererseits eventuell enthaltenes Fehlverhalten zu offenbaren. Da das Aktivitätsdiagramm in dieser ersten Phase noch sehr einfach ist, ergeben sich zu diesem Zeitpunkt durch die Ausführung des Testfallgenerators keine neuen Erkenntnisse und die erste Iteration der inneren Schleife kann abgeschlossen werden.

An dieser Stelle ist im Prozessablauf die Durchführung einer formalen Verifikation gegen Sicherheits- und Lebendigkeitsrandbedingungen vorgesehen. Allerdings stand zum Zeitpunkt der Modellierung (Anfang 2008) kein geeignetes formales Verifikationswerkzeug zur Verfügung, mit dem ein auf Aktivitätsdiagrammen basierendes Verhaltensmodell hätte verifiziert werden können. Daher konnte dieser Prozessschritt nicht durchgeführt werden und wurde übersprungen.

In den folgenden Iterationen wurden nun nach und nach die einzelnen Störungssituationen modelliert. Dies geschieht nach dem gleichen Vorgehen wie beim Regelablauf. So wurde beim vorliegenden Beispielmodell in der zweiten Iteration als Fehlerfall die Reaktion auf ein ausgefallenes Gelblicht am Bahnübergang modelliert: Sobald die Bahnübergangssteuerung einen Ausfall des Gelblichts[3] erkennt, muss sie sofort auf das Rotlicht umschalten und nicht erst nach der sonst üblichen Gelbzeit. Für die sich dadurch ergebenden Kommunikationsabläufe wird - wie beim Regelablauf - zunächst ein Sequenzdiagramm erstellt und anschließend das Verhaltensmodell so erweitert, dass auch der Fehlerfall korrekt abgebildet wird. Auch der nachfolgende Testzyklus folgt dabei dem zuvor beschriebenen Muster, verwendet als Eingangsdaten jedoch eine gegenüber der ersten Iteration vergrößerte Datenmenge. So muss sich das Verhaltensmodell nun gegen die Sequenzdiagramme von Regelfall (aus der ersten Iteration) und Gelblichtausfall (aus der aktuellen zweiten Iteration) bewähren, wodurch praktisch ein Regressionstest-Szenario entsteht. Sollte durch das Hinzufügen der Verhaltenskonstrukte in der zweiten Iteration das Gesamtverhalten ungewollt so verändert worden sein, dass das Sollverhalten aus der ersten Iteration nicht mehr erfüllt wird, würde dies jetzt bemerkt.

Auch die nachfolgende Testfallgenerierung stützt sich auf das nun erweiterte Verhaltensmodell. Mit dessen zunehmender Komplexität steigt auch die Wahrscheinlichkeit, dass durch die

[2] Ein ablauffähiges Modell erfordert minunter noch zusätzliche Modellierungsarbeiten (z.B. zur vereinfachten Abbildung der Systemumgebung), die aber nicht inhaltlich relevant sind und daher hier nicht näher beschrieben werden.

[3] Bei realen Anlagen im Bereich der Bahnen nach EBO wird eine bestimmte Anzahl an ausgefallenen Gelblichtern üblicherweise toleriert. Die Sicherungsanlage reagiert beispielsweise erst dann auf einen Gelblichtausfall, wenn es an einer bestimmten Anzahl an Signalgebern gleichzeitig zu einem Gelblichtausfall kommt.

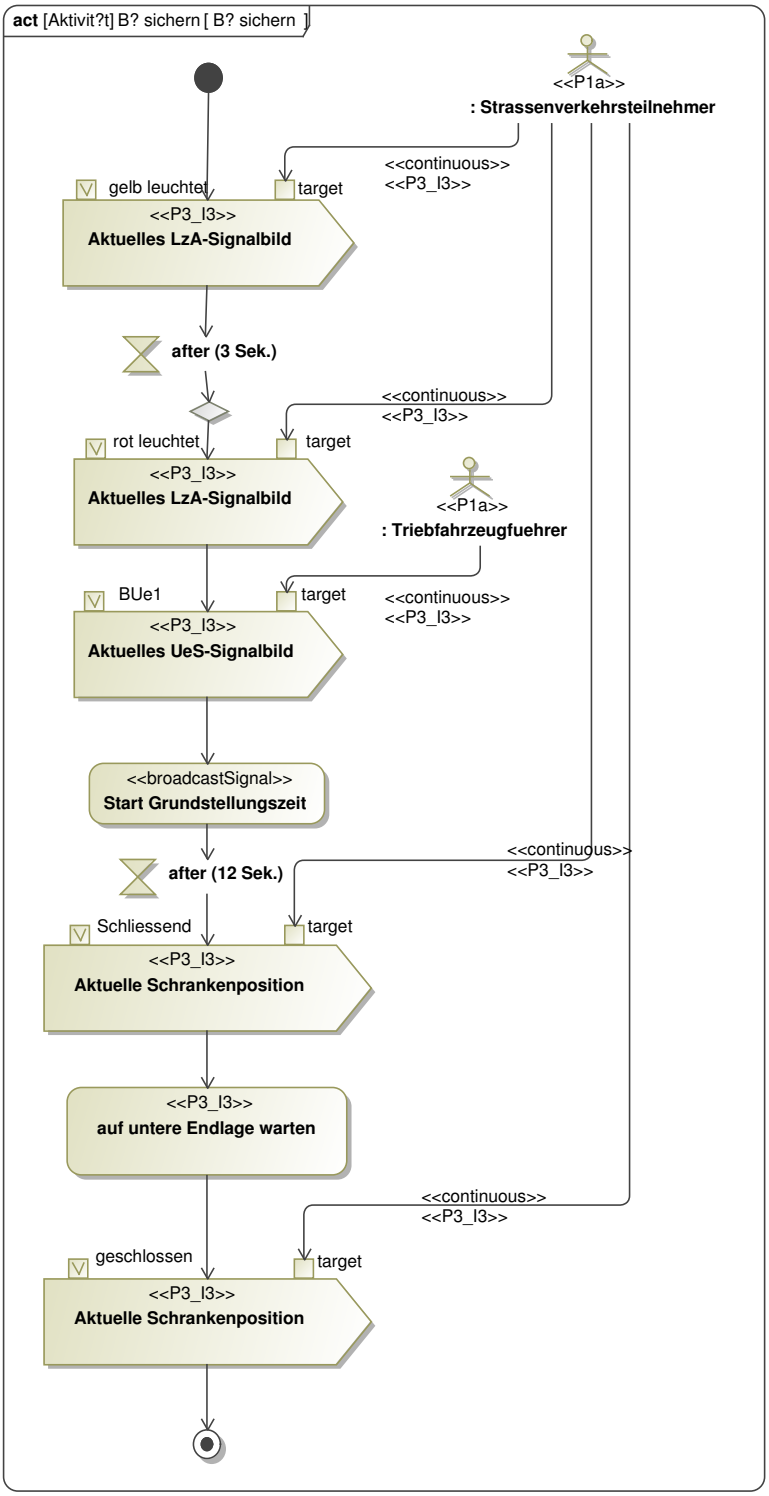


Abbildung 6.19.: Aktivitätsdiagramm für Anwendungsfall „BÜ sichern”

Testfallgenerierung ein bei der manuellen Ableitung der Sequenzdiagramme nicht berücksichtigtes Szenario erkannt wird. Im Beispielmodell geschah dies nach der 3. Iteration, in der die Reaktion auf ein ausgefallenes Rotlicht spezifiziert wurde. Bei der händischen Erstellung der Sequenzdiagramme wurde dabei nicht berücksichtigt, dass Gelblichtausfall und Rotlichtausfall auch gemeinsam auftreten können. Das für die korrekte Reaktion auf einen solchen Doppelausfall erforderliche Verhalten wurde bei der Erweiterung des Verhaltensmodells jedoch implizit mitmodelliert. Das Testfallgenerierungswerkzeug erzeugte nun aus dem Verhaltensmodell heraus ein Sequenzdiagramm für den Doppelausfall und machte das zugehörige korrekte Soll-Verhalten damit explizit. Damit war sichergestellt, dass es bei der weiteren Erweiterung des Verhaltensmodells nicht zu einer Regression kommen kann, in der das korrekte Verhalten bei einem Doppelausfall wieder verändert wird.

Für das Beispielmodell ergeben sich aus 12 Iterationen insgesamt 17 Szenarien für das Beispielmodell sowie ein in 13 Aktivitätsdiagrammen beschriebenes Systemverhalten. Gegenüber der vorherigen Phase 1 wurden bei der Erstellung der Diagramme auch die Angaben zu den ausgetauschten Informationen überarbeitet und präzisiert. So wird zwischen dem BÜ und dem Straßenverkehrsteilnehmer nicht mehr nur ein abstrakter Befahrbarkeitszustand ausgetauscht, sondern detailliertere Informationen zum Signalbild der LZA und zur Position der Schrankenbäume. Diese Modifikationen werden durch den Prozessablauf auch in allen anderen Modellsichten konsistent nachgezogen, was sich in der Systemabgrenzungssicht der Phase 2 (Bild 6.20) zeigt. Dort sind die gegenüber der Phase 1 modifizierten Signal-Namen zu erkennen.

Nach Abschluss der Phase 2 sind alle Systemfunktionen des Bahnübergangs im Beispielmodell enthalten und die Interaktionsszenarien sind nachweislich konsistent zum Systemverhalten.

6.3.4. P3 - Spezifikation der Subsystemarchitektur

In den vorhergehenden Phasen wurde das Modell soweit angereichert, dass es zu Beginn der Phase P3 ausführbar und testbar ist. Ziel der Phase 3 ist es nun, die Subsysteme des aktuellen SuB festzulegen, eine Subsystemarchitektur zu definieren und das global auf Systemebene definierte Verhalten auf die einzelnen Subsystemkomponenten aufzuteilen. Diese Subsystemkomponenten können dann in weiteren, rekursiven Anwendungen des hier beschriebenen Prozesses gemäß dem Ebenenkonzept weiter ausspezifiziert werden. Die Subsystemarchitektur wird in der Subsystemsicht dargestellt, die in Abschnitt 5.1.4.5 beschrieben wird. Die Durchführung dieser Phase ist dabei optional, da eine Subsystemdefinition nicht zwangsweise erfolgen muss.

Wird jedoch entschieden, diese durchzuführen, erfolgt auch die Abarbeitung dieser Aufgabe entsprechend dem Vorgehensmodell iterativ-inkrementell. Sinnvollerweise wird dabei über die einzelnen Subsystemkomponenten iteriert, d.h. pro Iteration wird eine Subsystemkomponente hinzugefügt. Dies geschieht durch den Subprozess S7, alle weiteren Subprozesse in dieser Phase entsprechen denen der Phase P2. Allerdings werden diese Aktivitäten in der Phase P3 zu einem einfacheren Ablauf verknüpft, der nur eine Iterationsschleife über die einzelnen Subsystemkomponenten enthält. Dieser Ablauf ist in Abbildung 6.21 dargestellt.

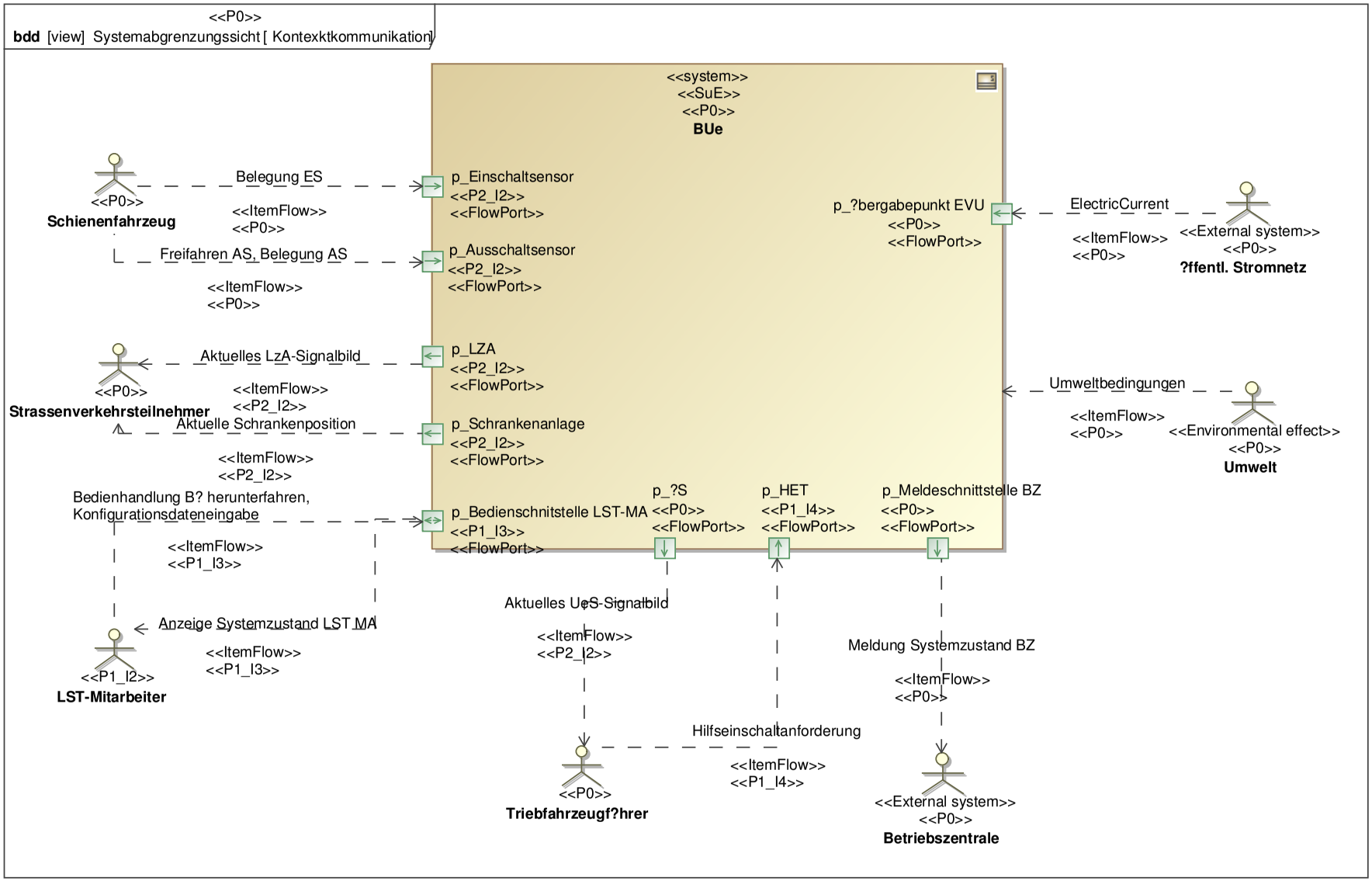


Abbildung 6.20.: Systemabgrenzungssicht nach Ende Phase 2 mit Modifikationen gegenüber Phase 1

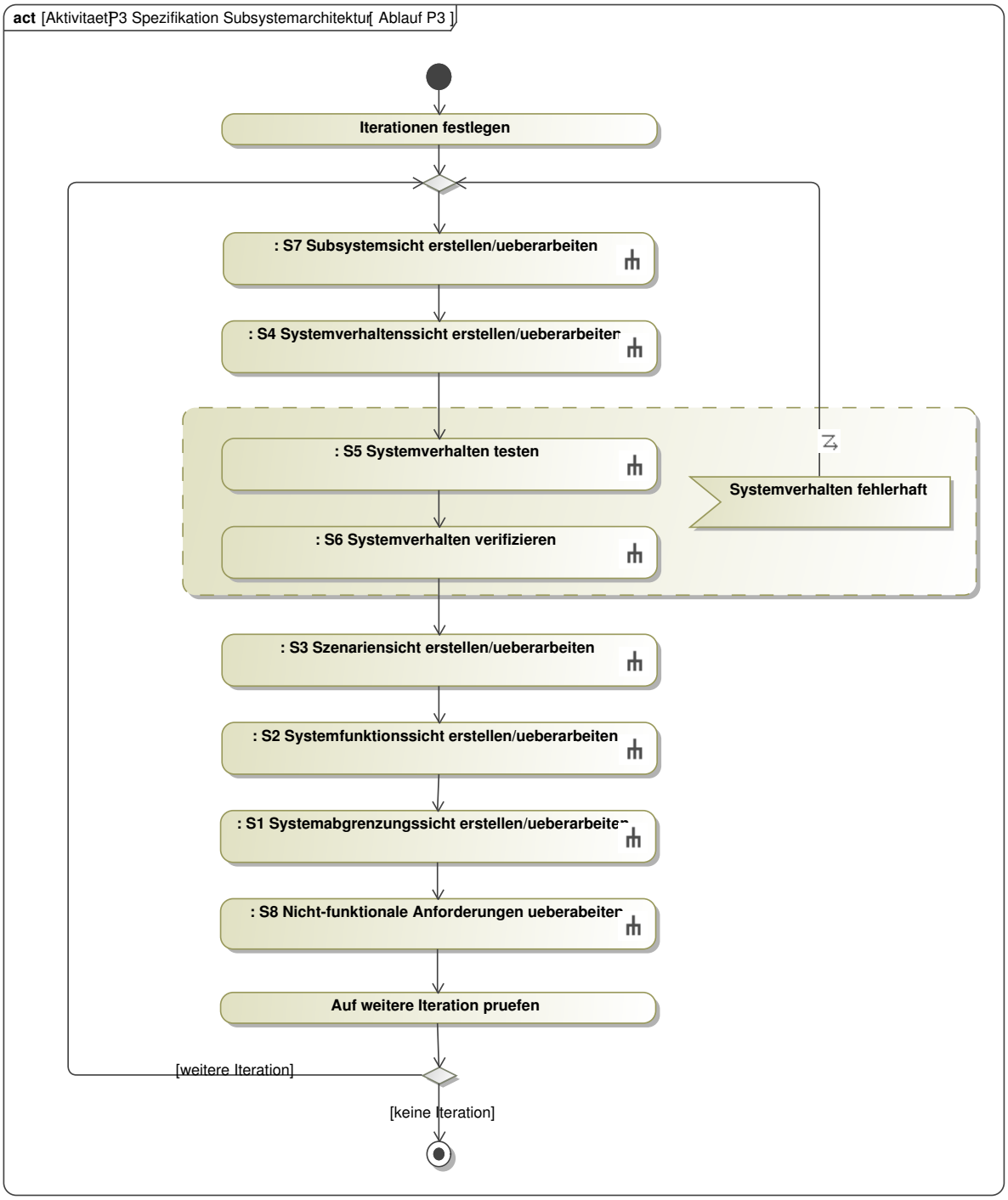


Abbildung 6.21.: Ablauf Phase P3

Welche Subsystemkomponenten überhaupt modelliert werden und wie diese voneinander abzugrenzen sind, lässt sich unabhängig von einem konkreten Anwendungsfall nicht vorgeben. Als allgemeine Hilfestellung lassen sich jedoch die folgenden Situationen anführen, in denen die Einführung einer eigenen Subsystemkomponente geboten ist:

- Die Komponente repräsentiert die Realisierung eines Ports aus der Systemabgrenzungssicht

- Die Komponente ist von besonderer Bedeutung für das Gesamtsystem, beispielsweise weil an sie herausragende Sicherheits- oder Leistungsfähigkeitsanforderungen gestellt werden
- Die Komponente repräsentiert ein physisch abgrenzbares Subsystem, dass beispielsweise einbaufertig von Drittherstellern geliefert wird (COTS - component off the shelf), oder vom Auftraggeber selbst bereitgestellt wird
- Die Komponente erfordert auf Grund ihrer Komplexität einen besonders hohen Spezifikationsaufwand in tieferen Modellebenen

Bild 5.21 in Kapitel 4 zeigt die Subsystemarchitektur des Beispielmodells. In der Überarbeitung der Verhaltenssicht in den Iterationen werden dann die einzelnen Verhaltenskonstrukte auf die neu hinzugefügten Subsystemkomponenten zugeteilt. Dies kann entweder grafisch durch Swimlanes erfolgen (siehe Bild 5.19) oder aber durch das Setzen entsprechender allocate-Abhängigkeiten.

Eine Zusammenstellung der wesentlichen Kriterien für den Ablauf dieser Phase findet sich in der nachfolgenden Tabelle 6.6.

Aspekt	
Iteration	In jedem Iterationsdurchlauf wird eine Subsystemkomponente hinzugefügt und in die Subsystemarchitektur integriert.
Iterationsendkriterium	Es kann keine weitere Subsystemkomponente ermittelt werden
Phasenendkriterium	Siehe oben

Tabelle 6.5.: Prozesskriterien Phase 3

6.3.4.1. Beispielmodell

Im Beispielmodell erfolgte die Zuteilung der einzelnen Funktionen auf die Subsystemkomponenten entsprechend den im vorherigen Abschnitt aufgeführten Hilfestellungen. So wurden für die Subsystemebene die folgenden Komponenten abgegrenzt, die eine Realisierung eines an der Systemgrenze befindlichen Ports darstellen:

- ein Ausschaltsensor als Instanz des Blocks „Induktionsschleife", der die Realisierung des Ports „p_Ausschaltsensor" darstellt
- zwei bis vier Einschaltsensoren „ES" als Instanzen des Blocks „Radsensor", als Realisierung des Ports „p_Einschaltsensor"
- vier bis sechs „Lichtzeichenanlagen" als Instanzen des Blocks „LZA", durch die der Port „p_LZA" realisiert wird
- zwei bis vier „Schrankenanlagen" als Instanzen des Blocks „Schrankenanlage", als Realisierung des Ports „p_Schrankenanlage"
- zwei bis vier Überwachungssignale „UeS als Instanzen des Blocks „UeS", zur Realisierung des Ports „p_UeS"
- zwei bis vier Hilfseinschalttaster „HET" als Instanzen des Blocks „HET", die den Port „p_HET" realisieren

Weiterhin wird im Beispielmodell angenommen, dass durch den Betreiber der Bahnübergangssicherung gewünscht wird, die Steuerung der Funktionen des Bahnübergangs durch eine zentrale BÜ-Steuerung zu realisieren. Diese Designvorgabe wird durch eine Komponente „BUe-Steuerung" repräsentiert, die im Diagramm der Subsystemarchitektur entsprechend dargestellt wird. Nachdem in der oben beschriebenen Weise die Systemarchitektur der nächst tieferen Ebene beschrieben wurde, wurde anschließend jedes Verhaltenskonstrukt einer dieser Komponenten zugeordnet. Bild 5.19 zeigt diese Zuordnung beispielhaft an einem Aktivitätsdiagramm.

6.3.5. P4 - Fortschreibung/Anpassung des Anforderungsmodells

Zu Beginn der Phase 4 sind entsprechend dem Konzept der Aufgabenorientierung alle Aufgaben aktiv, wodurch in jeder Iteration alle Subprozesse einmal durchlaufen werden. In dieser Phase entspricht der Prozessablauf einem klassischen iterativ-inkrementellen Prozess. Eine in dieser Phase an beliebiger Stelle eingepflegte Änderung im Modell verursacht eine Anpassung aller Sichten, bis das Modell wieder konsistent ist. Zudem führt jede Anpassung zu einem Test des Modellverhaltens und zu einem Verifikationslauf.

Diese Phase dient im Wesentlichen dem Einpflegen von Änderungen und der Lebenszyklus-Begleitung des fertigen Produkts. Sie berücksichtigt, dass auch nach abgeschlossener Subsystem-Architektur (aber noch während der Erstellung des Anforderungsmodells) die Möglichkeit für Änderungen und Ergänzungen am Modell gegeben sein muss. Zudem wird so auch die Tatsache berücksichtigt, dass in späteren Lebenszyklusphasen noch Modellanpassungen durchgeführt werden müssen. Soll beispielsweise während der Systementwicklung oder sogar am fertigen Produkt noch eine Änderung an der Funktionalität vorgenommen werden, ist es sinnvoll, diese Modifikationen auch im Anforderungsmodell zu berücksichtigen. Zum einen kann somit die Konsistenz zwischen Anforderungsmodell und realem System sichergestellt werden, zum anderen lassen sich die funktionalen Auswirkungen der Modifikationen durch die Test- und Verifikations-Subprozesse vor Implementierung im realen System erproben. Auf Grund der Lebenszyklus-begleitenden Funktion dieser Phase endet sie erst, wenn der Gesamt-Lebenszyklus des Systems beendet ist.

Je nach dem, ob eine Subsystem-Architektur durchgeführt wurde, oder nicht, entsprechen die Abläufe innerhalb der Phase entweder den Abläufen der Phase P2 (ohne Subsystemarchitektur) oder P3 (mit Subsystemarchitektur). Für die Beschreibung der Abläufe innerhalb der Phase wird daher auf die entsprechenden Unterkapitel verwiesen.

Aspekt	
Iteration	Für jede Änderung am Modell wird eine Iteration angesetzt. Die Art der Änderung bestimmt dabei die exakte Ausgestaltung der Iteration, diese wird dann entsprechend einer der Iterationskonzepte aus den Kapiteln 6.3.1 bis 6.3.4 ausgestaltet.
Iterationsendkriterium	Entspricht den Endkriterien des jeweils genutzten Iterationskonzeptes.
Phasenendkriterium	Ende des Systemlebenszyklus

Tabelle 6.6.: Prozesskriterien Phase 3

6.4. Subprozesse

Die eigentlichen Arbeitsschritte zur Erstellung der Sichten im funktionalen Modell sind in acht Subprozessen definiert. Dabei werden die Abläufe innerhalb der einzelnen Subprozesse durch SysML-Aktivitätsdiagramme modelliert. Die Modellierung erfolgt dabei so, dass hinreichend viel Freiraum für die Anpassung des Prozesses an die jeweiligen Gegebenheiten des Projekts bzw. des jeweiligen Anwenders besteht. Die innerhalb der Subprozesse beschriebenen Aktivitäten sind daher eine Art Platzhalter, die bei der Prozessanpassung (dem so genannten *Tailoring*) dann durch konkrete Aktivitäten ersetzt werden müssen. Weiterhin ist nicht zwingend vorgegeben, in welcher Reihenfolge die einzelnen Aktivitäten innerhalb der Subprozesse ausgeführt werden. Vielmehr muss je nach Kontext der Subprozessausführung eine geeignet erscheinende Arbeitsweise durch den Modellierer selbst definiert werden. Aus diesem Grund wird der Arbeitsfluss innerhalb der Subprozesse durch parallele Aktivitäten dargestellt, wodurch diese Freiheit repräsentiert wird. Ein weiterer Freiheitsgrad ergibt sich aus der Tatsache, dass die in einer konkreten Situation tatsächlich in den einzelnen Aktivitäten durchgeführten Arbeitsschritte vom Iterationskonzept der übergeordneten Phase abhängig sind. Dabei legt die Phasendefinition fest, ob im Subprozess lediglich bestehende Modellartefakte angepasst oder zusätzlich neue Elemente ins Modell eingeführt werden sollen. Dies zeigt sich auch in den entsprechenden Bezeichnungen der einzelnen Aktivitäten innerhalb der Subprozesse in der Form „Artefakt erstellen/überarbeiten". Eine striktere Definition mit entsprechenden Fallunterscheidungen als Verzweigungen im Kontrollfluss wäre zwar möglich gewesen, wurden aber aus Gründen der Verständlichkeit und Übersichtlichkeit nicht weiter verfolgt.

Die Subprozesse S1 bis S4 und S7 beziehen sich auf die Erstellung des funktionalen Modells und ähneln sich in ihrer grundlegenden Struktur: In ihnen werden parallel mehrere *Aktivitäten* ausgeführt, die die Modellelemente der zugehörigen Sicht bearbeiten. Die Anzahl der Aktivitäten richtet sich dabei nach der Anzahl der Modellelemente in der Sicht. So werden im Subprozess S1 beispielsweise fünf Aktivitäten ausgeführt, von der jede eins der fünf Modellelemente (SuB, Akteure, Kommunikationsflüsse, Portdefinitionen, Signale) der Systemabgrenzungssicht (siehe Abschnitt 5.1.4.1) modifiziert. Die Modellelemente selbst (siehe Abschnitt 5.1.2) werden im Prozess-Metamodell durch je einen persistenten *DataStoreNode* [OMG07-2, S. 358] repräsentiert und sind über eine Datenfluss-Beziehung mit der Aktivität verbunden. Der DataStoreNode macht dabei deutlich, dass die Modifikationen über die aktuelle Ausführung des Subprozesses hinaus dauerhaft erhalten bleibt. Die einzelnen DataStores stehen dabei für die Menge aller Modellelemente eines Typs. So repräsentiert beispielsweise der DataStore „Menge der Port-Definitionen" *alle* Port-Defnitionen im Teilmodell.

Die Subprozesse S5 (Systemverhalten testen) und S6 (Systemverhalten verifizieren) weisen auf Grund ihrer prinzipiell anderen Zielrichtung eine zu den anderen Subprozessen abweichende Struktur auf. Ebenfalls eine Sonderstellung nimmt der Subprozess S8 ein. Dieser dient der Verwaltung der nicht-funktionalen Anteile der Anforderungsspezifikation und stellt die Verknüpfungen mit dem funktionalen Modell her.

6.4.1. S1 - Systemabgrenzungssicht bearbeiten

Im Subprozess S1 sind diejenigen Aktivitäten definiert, die zur Überarbeitung der Systemabgrenzungssicht erforderlich sind. Zunächst wird das SuB erstellt bzw. überarbeitet, dann erfolgt die Erstellung/Überarbeitung der Akteure, Portdefinitionen, Informationsflüsse und Signale sowie eine permanente Überprüfung der Konsistenzkriterien.

Das Aktivitätsdiagramm für die Abläufe im Subprozess findet sich in Bild B.5 im Anhang.

6.4.2. S2 - Systemfunktionssicht bearbeiten

Der Subprozess S2 beschreibt diejenigen Aktivitäten zur Überarbeitung der Systemfunktionssicht. Der Subprozess beginnt mit einer Analyse der bestehenden Anwendungsfälle (in der Aktivität „vorhandene Anwendungsfälle analysieren"). Welche Modellelemente und -strukturen in diese Analyse einbezogen werden und wie diese Prüfung abläuft, ist dabei nicht exakt vorgegeben, da diese je nach Anwendungsszenario des Subprozesses unterschiedlich ausgestaltet werden muss und somit vom Modellierer zu definieren ist. Bei der hierarchischen Ableitung der Systemfunktionen entsprechend Phase P2 (siehe 6.4) werden hier beispielsweise die Aktivitätsdiagramme bereits vorhandener Anwendungsfälle analysiert, eine Aktivität ausgewählt und diese dann im weiteren Ablauf des Subprozesses als Anwendungsfall modelliert.

Anschließend werden im Subprozess durch die entsprechenden Aktivitäten Anwendungsfälle, Akteure, Informationsflüsse und Signale erstellt bzw. überarbeitet. Auch in diesem Subprozess haben die einzelnen Aktivitäten Platzhaltercharakter und lassen Raum für Anpassungen an die jeweilige konkrete Implementierung des Prozesses. In der Aktivität „Anwendungsfälle erstellen/-überarbeiten" werden in der konkreten Verwendung dieses Subprozesses beispielsweise auch die Aktivitätsdiagramme zur Ableitung weitere Systemfunktionen entsprechend dem in Abschnitt 6.3.2 beschriebenen Vorgehen erstellt.

Die Definition des Subprozesses als Aktivitätsdiagramm zeigt Bild B.6 im Anhang.

6.4.3. S3 - Szenariensicht bearbeiten

Im Subprozess S3 sind alle Aktivitäten definiert, die zur Erstellung der Szenariensicht erforderlich sind. Üblicherweise wird dieser Subprozess so aus der übergeordneten Phase heraus ausgeführt, dass pro Ausführung genau eine Interaktion erstellt wird. Dies erfolgt unmittelbar zu Beginn des Subprozesses, anschließend werden in einer Reihe paralleler Aktivitäten alle benötigten Inhaltselemente erstellt bzw. modifiziert. Dies sind:

- SuB und Akteure (jeweils durch LifeLines repräsentiert)
- Nachrichten
- Signale
- Kontrollstrukturen

Auch für diesen Subprozess existiert eine semi-formale Definition als Aktivitätsdiagramm. Diese ist in Abbildung B.7 im Anhang dargestellt.

6.4.4. S4 - Verhaltenssicht bearbeiten

Der Subprozess S4 umfasst alle Aktivitäten zur Modellierung des Systemverhaltens. Dies umfasst die Erstellung der Aktivitätsdiagramme bzw. Zustandsmaschinen, die Erstellung der entsprechenden Verhaltenskonstrukte in diesen Diagrammen und die ggf. erforderlichen Modifikationen an Zuteilungen, Akteuren, Signalen und Portdefinitionen. Dabei werden die zugehörigen Inhaltselemente des Modells modifiziert, wie in Bild 6.22 in der Prozessdefinition des Subprozesses zu erkennen ist.

Entsprechend dem Phasenkonzept wird dieser Subprozess bei jeder Änderung an der Verhaltenssicht aufgerufen, so beispielsweise in der Phase P2 für jedes Interaktionsszenario im funktionalen Modell. Modellierungsaufgabe im Subprozess ist jeweils die Anpassung des Verhaltensmodells an die Verhaltensvorgaben aus der zu diesem Zeitpunkt bestehenden Gesamtmenge aller

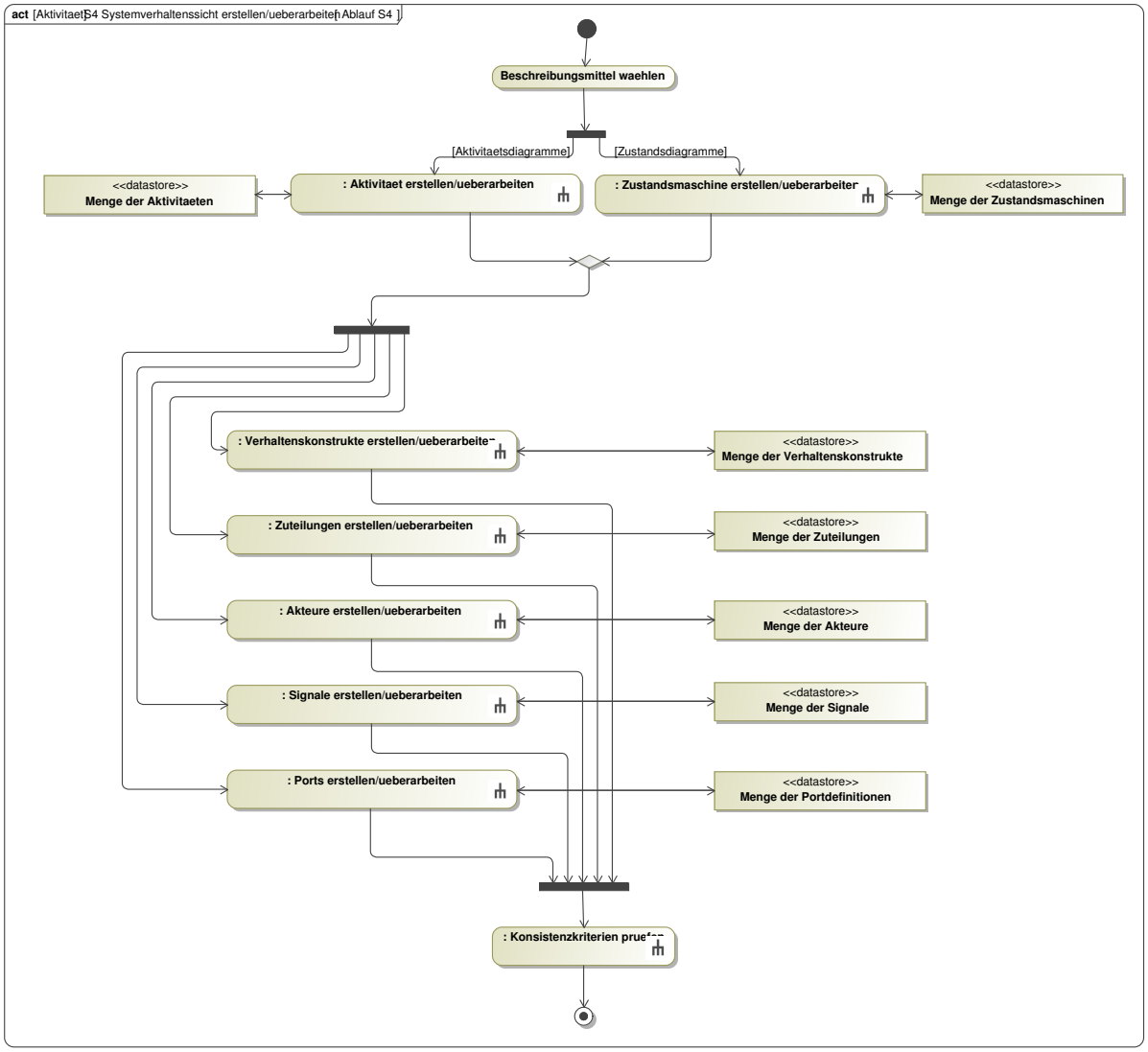


Abbildung 6.22.: Prozessablauf des Subprozesses S4

Szenarien. Die Umsetzung dieser Modellierungsaufgabe erfordert daher anteilig sowohl die Modifikation bestehender Verhaltens-bestimmender Artefakte, aber auch die Ergänzung durch neue Modellelemente.

Üblicherweise werden die im Fokus dieses Subprozesses stehenden Verhaltens-definierenden Konstrukte manuell erstellt. Zurzeit befindet sich jedoch eine interessante, teil-automatisierbare Möglichkeit zur Modellierung stark Logik-zentrierter Problemstellungen in der Entwicklung. Die Idee ist, die Aussagen von Entscheidungstabellen in aussagenlogische Formeln zu übersetzen. Hon beschreibt diesen Ansatz in [HGE08], der im Folgenden kurz vorgestellt wird.

Entscheidungstabellen sind ein Werkzeug zur strukturierten Darstellung komplexer Regelwerke, die aus der Verknüpfung zahlreicher logischer Bedingungen bestehen [MEY75]. Ein gutes Beispiel für solche komplexen Regelwerke im Eisenbahnwesen sind beispielsweise die Prüfbedingungen zur Zulassung von Fahrstraßen. Die manuelle Erstellung von Aktivitäts- oder Zustandsautomaten, die diese Prüfungen im Anforderungsmodell implementieren ist aufwändig und fehleranfällig. Daher bietet sich für diese Art von Problemstellungen das im Folgenden be-

schriebene Vorgehen an:

- Zunächst werden die Entscheidungsregeln für das logische Problem in Form von Entscheidungstabellen erfasst. Diese Tabellen können anschließend automatisiert auf Vollständigkeit und Konsistenz geprüft werden.
- Der in der Entscheidungstabelle hinterlegte Informationsgehalt kann dann in aussagenlogische Formeln übersetzt werden. Diese Formeln stellen eine Funktion dar, die basierend auf den logischen Abhängigkeiten der Entscheidungstabellen und dem aktuellen Wert von Variablen entweder „wahr" oder „falsch" zurückliefert. Die einzelnen Variablen werden dabei durch die Bedingungen im allgemeinen Bedingungsteil [MEY75, S. 19] der Entscheidungstabelle repräsentiert.
- Da die aussagenlogischen Formeln boolesche Werte zurückliefern, können sie im Verhaltensmodell beispielsweise als *guard*-Ausdrücke [OMG07-2, S. 325] für Kontrollflusskanten verwendet werden. Als Variablen in der aussagenlogischen Formel lassen sich die Attribute eines Objekts verwenden, wodurch das Ergebnis der Auswertung des *guards* vom momentanen Wert der Attribute des Objektes abhängig wird. Durch dieses Konstrukt lässt sich der Kontrollfluss innerhalb von Aktivitätsdiagrammen durch den Informationsgehalt der Entscheidungstabellen steuern.

Das beschriebene Vorgehen wurde bislang noch nicht in der Praxis erprobt und seine Anwendbarkeit sollte zunächst an Fallstudien verifiziert werden. Es handelt sich dabei jedoch um einen vielversprechenden Ansatz, um komplexe, logische Entscheidungsregeln möglichst einfach in ein objektorientiertes Anforderungsmodell zu integrieren.

6.4.5. S5 - Systemverhalten testen

Generell kommt dem Testen insbesondere bei sicherheitskritischen Systemen während des gesamten Entwicklungsprozesses eine besondere Bedeutung zu, da durch systematisches Testen eine hohe Erkennungsrate von Fehlern erzielt werden kann. Es ist daher sinnvoll, mit dem Testen möglichst früh im Lebenszyklus des Systems zu beginnen. Erklärtes Ziel des in dieser Arbeit beschriebenen Konzeptes ist aus diesem Grund, Testmethodiken bereits auf die Anforderungsspezifikation anzuwenden. Dieses Konzept wird im Subprozess S5 umgesetzt und in den folgenden Unterkapiteln beschrieben.

Der Subprozess S5 nimmt im gesamten Prozessablauf somit eine herausgehobene Stellung ein, da der Test des funktionalen Modells eine der wesentlichen neuen Möglichkeiten ist, die sich aus der Modellierung von Anforderungen gegenüber rein textlicher Beschreibung ergeben. Weiterhin wird durch diesen Subprozess sichergestellt, dass die Szenariensicht und das modellierte Systemverhalten zueinander konsistent sind. Durch die Integration automatisch generierter Testfälle werden darüber hinaus die händisch erstellten Szenarien ergänzt. Somit erfolgt innerhalb dieses Subprozesses nicht nur die Durchführung der Tests, sondern auch eine Optimierung und Ergänzung der Testbasis.

Weil diese Testbasis die Systemebene abdeckt, kann sie zudem als Grundlage für einen späteren Systemintegrationstest bzw. Abnahmetest des realen Systems dienen. So werden beispielsweise im Lokbau bei der Fahrzeugabnahme momentan Validierungs- und Testpläne verwendet, die aus den natürlich-sprachlichen Anforderungsspezifikationen händisch abgeleitet werden. Würde jedoch ein funktionales Modell im Sinne dieser Arbeit existieren, könnten die erforderlichen Testfälle direkt aus diesem Modell abgeleitet werden. Dies würde den Aufwand zur Testfallerstellung reduzieren und die Qualität der Testfallabdeckung transparenter machen. Daher dient

das Testen nicht nur der Qualitätssicherung des Anforderungsmodells an sich, sondern auch anderen Prozessschritten im Gesamtlebenszyklus.

Festzuhalten ist allerdings, dass durch Testen nie das gesamte Verhalten des Systems geprüft werden kann. Testen ist prinzipiell eine Falsifizierungs- und keine Verifizierungsmethode. Durch Testfälle kann zwar die Inkorrektheit eines Systems nachgewiesen werden (nämlich dann, wenn das Ergebnis eines Tests die Spezifikation verletzt), nicht aber die Korrektheit (da immer noch ein Fehler im nicht von den Testfällen abgedeckten Verhaltensbereich vorhanden sein kann). Jedoch steigt durch eine gute Testfallabdeckung die Wahrscheinlichkeit, die Zahl der Fehler in der Anforderungsspezifikation zu verringern.

6.4.5.1. Ziele des Testens in diesem Subprozess

Um eine geeignete Teststrategie und entsprechende Werkzeuge auswählen zu können, werden in diesem Unterabschnitt anhand der in Abschnitt 6.3.3.4 beschriebenen Grundlagen die Anforderungen an den Test-Subprozess formuliert. In Abschnitt 6.4.5.3 wird anschließend das daraufhin gewählte Vorgehen beschrieben.

Für das Testen im Rahmen des in dieser Arbeit beschriebenen Prozesses lassen sich zwei wesentliche Ziele definieren:

- Ziel A: Prüfung der Verhaltenssicht auf Konsistenz mit der Szenariensicht („Erfüllt mein Verhaltensmodell die Betriebsszenarien?").
- Ziel B: Hilfestellung bei der Ableitung einer möglichst vollständigen Szenariensicht („Wie kann ich die Testbasis verbreitern? Habe ich implizit mehr modelliert, als in den Szenarien definiert ist?")

Beide Ziele sollen dabei möglichst direkt auf Basis des momentanen Arbeitsstandes des funktionalen Modells erreicht werden, also ohne dass ein Bruch im Beschreibungsmittel, der Methodik oder der Werkzeuge erforderlich wird. Daraus lässt sich ableiten, dass auch das Testverfahren *modellbasiert* sein muss. Üblicherweise wird unter diesem Begriff verstanden, dass zur Ableitung von Testfällen eine modellhafte Beschreibung der Umwelt oder des idealisierten Systemverhaltens verwendet wird [UPL06, S. 1]. Die aus diesem Modell beispielsweise durch ein Testfallgenerierungswerkzeug erstellten Testfälle können dann auf das zu testende Artefakt (z.B. eine Steuersoftware als konkrete Realisierung des Systemverhaltens) angewendet werden. Üblicherweise sind Testmodell und zu testendes Artefakt dabei nicht identisch und können durchaus verschiedenen Lebenszyklusphasen angehören: Ein Testmodell aus der Phase der Systemanforderungen kann beispielsweise zum Testen der - viel später erfolgenden - Systemintegration verwendet werden.

Im Rahmen dieser Arbeit soll die Bedeutung des modellbasierten Testens dahingehend ausgedehnt werden, dass das Anforderungsmodell sowohl der Ableitung der Testfälle dient, als auch Testgegenstand gleichermaßen ist. Das Testen erfolgt hier also *innerhalb* ein- und desselben Modells. Allerdings nehmen verschiedene Modellteile unterschiedliche Rollen ein: Die Szenariensicht repräsentiert die Testfälle, die Verhaltenssicht den Testgegenstand. Durch dieses Vorgehen soll überprüft werden, ob das in der Verhaltenssicht modellierte Verhalten die Vorgaben der Szenarien erfüllt. Durch Ziel A soll somit die Konsistenz beider Modellelemente sichergestellt werden.

Ziel B lässt sich dahingehend konkretisieren, dass es möglich sein soll, die Elemente der Verhaltenssicht derart zu analysieren, dass sich daraus Szenarien entsprechend der Szenariensicht ableiten lassen. Dies ist prinzipiell möglich, da die Verhaltenssicht inhaltlich immer „reicher" als

die Szenariensicht ist: Letztere zeigt nur beispielhafte Ausschnitte des tatsächlich modellierten Verhaltens. Ziel B zielt zudem darauf ab, implizite Annahmen während der Verhaltensmodellierung explizit zu machen und auf Konsistenz mit dem vorgesehenen Verhalten des Systems zu prüfen. In der Praxis wurden bei der Erstellung des Beispielmodells Fehlerszenarien für einen „Fadenfehler Gelblicht" und einen „Fadenfehler Rotlicht" der Lichtzeichenanlage spezifiziert. In der Verhaltenssicht war implizit jedoch auch das (korrekte) Verhalten für einen kombinierten Fehlerfall „Fadenfehler Gelb- und Rotlicht" enthalten. Für diesen Fehlerfall wurde durch die Analyse der Verhaltenssicht ein entsprechendes Sequenzdiagramm ermittelt, wodurch die zuvor nur implizit vorliegende Spezifikation als Szenario explizit gemacht wurde.

Anzumerken ist, dass das beschriebene Vorgehen dann problematisch sein kann, wenn die Szenariensicht (also die Testfälle) und die Verhaltenssicht (also der Testling) von der gleichen Person modelliert werden. Die Gefahr besteht, dass sich ein prinzipieller Denkfehler in beiden Sichten reproduziert und somit ein eigentlich falsches Verhalten als „korrekt" getestet wird. Auch wenn innerhalb dieser Arbeit prinzipiell keine Aussagen über Rollen und personelle Abhängigkeiten getroffen werden, ist es höchst sinnvoll, zumindest die Szenariensicht und die Verhaltenssicht von zwei unterschiedlichen Personen bearbeiten zu lassen. Es ergeben sich somit für den mikroskopischen Bereich der Anforderungserstellung zwei Rollen, die der des „Implementierers" und der des „Testers" bei Softwareentwicklungsprozessesn entsprechen.

Um die genauen Anforderungen an diesen Prozessschritt genauer fassen zu können, werden im folgenden Unterabschnitt 6.4.5.2 die benötigten Eigenschaften für die Testfallgenerierung mittels einer Taxonomie genauer klassifiziert.

6.4.5.2. Klassifizierung des Verfahrens zur Testfallgenerierung

Um das benötigte Verfahren zur Testfallgenerierung genauer definieren zu können, sollen die Anforderungen aus dem vorherigen Abschnitt gegen eine Taxonomie möglicher Verfahren evaluiert werden. Diese Taxonomie wurde von Utting, Pretschner und Legeard in [UPL06] vorgestellt. Sie unterscheiden dabei sieben Merkmale mit verschiedenen Ausprägungen, mit denen modellbasierte Testsfallgenerierungsverfahren differenziert werden können. Im Folgenden werden die für diese Arbeit relevanten Merkmale kurz vorgestellt und dabei diejenige Merkmalsausprägung ausgewählt, die ein möglichst gut angepasstes Testfallgenerierungsverfahren beschreibt.

(a) *Gegenstand des Modells* [UPL06, S. 4/5] - Dieses Merkmal beschreibt, ob das Modell eher das Verhalten des zu testenden Systems darstellt oder eher das Verhalten der Systemumgebung. Diese Differenzierung ist entscheidend für die Strategie der Testfallgenerierung. Für das hier angewendete und oben beschriebene Vorgehen zur Testfallerzeugung ist das Verhalten des Systems maßgeblich. Das gewählte Werkzeug muss also die Analyse des Systemmodells unterstützen.

(b) *Modellredundanz* [UPL06, S. 5/6] - hierbei wird zwischen dem Fall separater Modelle für Implementierung und Testfallgenerierung und dem Fall eines gemeinsamen Modells für beide Aufgaben unterschieden. Im vorherigen Abschnitt wurde dargelegt, dass die Tests innerhalb eines Modells durchgeführt werden sollen. Demnach liegt hier der Fall eines gemeinsamen Modells vor, den das Werkzeug beherrschen können muss.

(c) *Modellcharakteristik* [UPL06, S. 6] - das Merkmal dient der Unterscheidung der Charakteristiken eines Modells im Bezug auf Determinismus seiner Ausgaben, der Berücksichtigung von Echtzeitanforderungen sowie der Unterscheidung von diskreten und kontinuierlichen Systemen. Für Anforderungsmodelle gilt dabei, dass sie an der Systemgrenze üblicherweise ein deterministisches Verhalten aufweisen. Die Berücksichtigung harter Echtzeitan-

forderungen bei der Testfallgenerierung ist wünschenswert, aber momentan noch Ziel umfangreicher Forschung und für den Produktionseinsatz noch nicht verfügbar. Ähnliches gilt auch für kontinuierliche Systeme: Teststrategien für kontinuierliche Systeme wären wünschenswert (insbesondere, da die SysML derartige Konstrukte explizit unterstützt), sind aber noch nicht in Form fertiger Produkte verfügbar.

(d) *Modellierungsparadigma* [UPL06, S. 7] - durch dieses Merkmal wird festgelegt, durch welchen Modellierungsansatz das Modell erstellt wurde. Im Rahmen des zitierten Papiers fallen UML und SysML-Modell in die Kategorie der Transitions-basierten Notationen (Zustandsmaschinen) bzw. Datenfluss-basierten Notationen (Aktivitätsdiagramme).

(e) *Testfallgenerierung* [UPL06, S. 8/9] - hierbei wird unterschieden, wie Testfälle aus dem Modell abgeleitet werden. Relevant für die hier angestellten Betrachtungen ist dabei nur die Unterscheidung in manuelle und automatische Testfallgenerierung. Zur Erfüllung von Ziel B muss das Werkzeug die automatische Generierung von Testfällen unterstützen.

6.4.5.3. Verfahren und Werkzeuge

Aus den Zielen in Abschnitt 6.4.5.1 und Taxonomie für die Testfallgenerierung im vorherigen Abschnitt 6.4.5.2 lassen sich das im Folgenden beschriebene, prinzipielle Verfahren und Aussagen zu den benötigten Werkzeugen ableiten.

Die Umsetzung von Ziel A ist unabhängig von der vorgestellten Taxonomie und erfolgt durch die Verwendung eines modellbasierten Testausführungswerkzeuges. Dieses soll das Freischneiden des Systems entlang der in der Systemabgrenzungssicht definierten Systemgrenze, die Stimulierung des Systems mit den Eingabestimuli aus den Szenarien und die Aufzeichnung der Reaktionen des Systems ermöglichen. Ein Beispiel für ein geeignetes Werkzeug für UML/SysML-Umgebungen ist der *TestConductor* der OSC AG, der sich in IBM/telelogic Rhapsody integrieren lässt [OSC08-1] und der im Rahmen dieser Arbeit verwendet wurde. Bei entsprechender sorgfältiger Erstellung der Szenariensicht[4] können deren Diagramme direkt als Testfälle verwendet werden. Auch die Testausführung erfolgt weitgehend automatisiert: Aus dem ausführbaren Modell wird Quellcode erzeugt, dieser wird compiliert, ausgeführt und das Modell daraufhin den in den Testfällen definierten Stimuli (in Form der auf das SuB zulaufenden Nachrichten) unterworfen. Die Antworten des Modells werden vom TestConductor aufgezeichnet und mit den in den Szenarien vorgegebenen Soll-Ausgaben (entsprechen den aus dem SuB herausströmenden Nachrichten) verglichen. Das Ergebnis dieses Vergleichs bestimmt anschließend das Urteil darüber, ob der Testfall bestanden wurde oder nicht.

Die Erreichung von Ziel B erfordert eine Analyse des tatsächlich vorliegenden Modellverhaltens und eine darauf basierende automatische Testfallerstellung. Dabei muss das verwendete Werkzeug die in 6.4.5.2 beschriebenen Merkmale erfüllen. Ein Beispiel für ein entsprechendes Werkzeug ist *ATG* der OSC AG [OSC08-2], das sich, wie auch *TestConductor*, in IBM/telelogic Rhapsody integrieren lässt. ATG ermöglicht die Analyse von UML/SysML-Modellen des zukünftigen Systems (erfüllt Merkmal (a)), deren Verhalten in Form von Zustandsmaschinen oder Aktivitätsdiagrammen definiert ist (erfüllt Merkmal (d)) und erzeugt anhand dieser Analyse innerhalb des Modells Szenarien, die als Testfälle verwendet werden können (erfüllt Merkmal (b) und (e)). Die hinter ATG stehende Algorithmik (die Grundlagen sind in [LET05] beschrieben) versucht, für ein gegebenes Modell Sequenzen von Eingangsstimuli zu finden, die alle Zustände, Transitionen, Ereignisaufrufe und Operationen in den Verhaltensdiagrammen anfahren. Ergebnis des ATG-Laufs ist eine Menge an Szenarien, die in ihrer Charakteristik den manuell erstellten

[4] Dies betrifft insbesondere die Verwendung korrekter Namen für Signale und Argumente

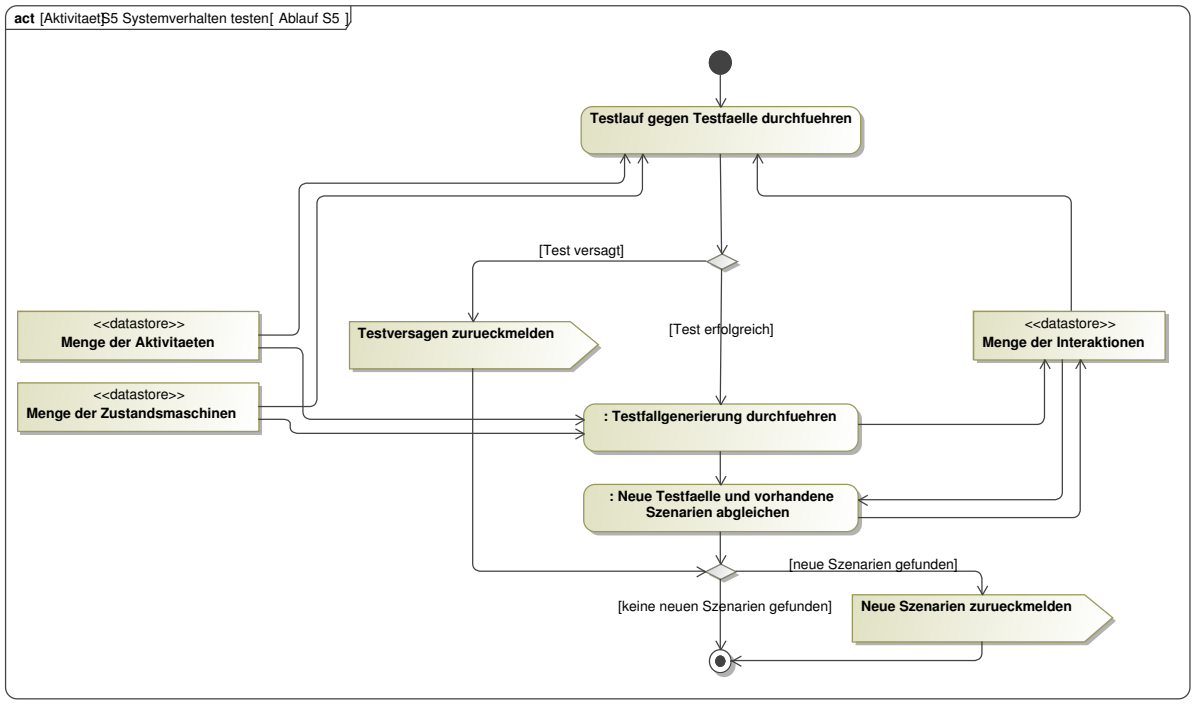


Abbildung 6.23.: Aktivitätsdiagramm Subprozess S5

Szenarien entsprechen. Das Testfallgenerierungswerkzeug vermag dabei natürlich keine fachliche Einordnung der gefundenen Szenarien vorzunehmen, wodurch sich in deren Gesamtmenge üblicherweise einige unbrauchbare Sequenzen finden:

- Triviale Szenarien mit nur einer eingehenden Nachricht, die unmittelbar zum Erreichen des ersten Zustands des Modells führt. Ein solches Szenario erscheint aus Sicht von ATG sinnvoll, da damit z.B. ein noch fehlender Zustand angefahren werden konnte. Aus Sicht des Systemtests ist ein solches Szenario jedoch nicht aussagekräftig.
- Szenarien, die Teilmengen von ausführlicheren Szenarien sind. Diese Szenarien enden, obwohl ein realer betrieblicher Vorgang noch nicht abgeschlossen ist. Diese Szenarien sind zwar aussagekräftig als Testfall, sollten aber beispielsweise durch Vereinigung mit anderen Szenarien auch in einen geeigneten betrieblichen Kontext gebracht werden.

Aus diesem Grund muss im Prozess eine Aktivität vorgesehen werden, die die automatisch generierten Testfälle einem inhaltlichen Review unterzieht. Diese Eigenschaft reduziert den oftmals angeführten Zeitvorteil der automatischen Testfallgenerierung, da zwar Arbeitsaufwand bei der Erstellung der Testfälle eingespart wird, im Gegenzug aber zusätzlicher Aufwand entsteht, um die generierten Testfälle zu reviewen. Unabhängig davon bleiben jedoch der Vorteil der automatisierten Analyse des Verhaltensmodells und die damit verbundene Aussicht auf eine höhere Testfallabdeckung bestehen.

6.4.5.4. Vorgehen

Das für den Subprozess S5 gewählte Vorgehen kann dem Aktivitätsdiagramm in Abbildung 6.23 entnommen werden.

Der Subprozess beginnt mit der Aktivität „Testlauf gegen Testfälle durchführen” und setzt damit Testziel A um. Wurde hierbei festgestellt, dass das modellierte Verhalten nicht den in den Szenarien spezifizierten Systemreaktionen entspricht, wird der Subprozess beendet. In der übergeordneten Phase muss dann die Inkonsistenz zwischen Szenarien und Verhaltensmodell behoben werden.

War der Test jedoch erfolgreich, wird auf Basis des gesamten bis zum momentanen Zeitpunkt modellierten Verhaltens die Testfallgenerierung angestoßen. Die dabei erzeugten Szenarien werden einem Review unterzogen und mit der bereits bestehenden Menge an Szenarien abgeglichen. Sollten sich dabei neue Szenarien ergeben, wird dies ebenfalls der übergeordneten Phase angezeigt und in deren weiteren Abläufen berücksichtigt. Diese Aktivität setzt damit das Testziel B um.

Der Review der generierten Szenarien und deren Integration in die bestehende Menge an Szenarien erfolgt dabei zurzeit durch einen rein manuellen Vergleich der einzelnen Szenarien. Dies ist für größere Mengen an Testfällen, wie sie bereits bei mäßig komplexen Systemen entstehen können, kaum praktikabel. Daher sollte in weiteren Forschungsarbeiten eine Möglichkeit zum teilautomatisierten Abgleich zwischen neu generierten Testfällen und der bereits existierenden Testfallbasis erörtert werden. Als erster Schritt könnte ein automatischer Abgleich eines neuen Szenarios mit den bereits bestehenden Szenarien über eine einfache Vergleichsfunktion durchgeführt werden. Damit könnte schneller erkannt werden, ob ein durch die Testfallgenerierung ermitteltes Szenario bereits in der Menge der Szenarien enthalten ist oder nicht. Eine mögliche Implementierung zeigt die Funktion „Sequence Diagram Compare” von IBM/telelogic Rhapsody, die zwei Sequenzdiagramme vergleicht und das Ergebnis durch farbige Markierungen darstellt. Abbildung 6.24 zeigt das Ergebnis eines solchen Vergleichs. Eine mögliche Weiterentwicklung dieser Funktion könnte dann automatisiert größere Mengen von Sequenzdiagrammen vergleichen und Unterschiede entsprechend melden.

6.4.6. S6 - Systemverhalten verifizieren

Eine weitere Möglichkeit zur Prüfung des Systemverhaltens besteht in der Anwendung eines Model-Checkers zur formalen Verifikation des Verhaltensmodells. Dabei wird das Verhalten eines Systems gegen eine Spezifikation verglichen, die aus einzelnen Sicherheits- oder Lebendigkeitsbedingungen besteht. Sicherheitsbedingungen beschreiben dabei Systemzustände, die vom System niemals eingenommen werden dürfen. Lebendigkeitsbedingungen prüfen hingegen, ob das System einen bestimmten - beispielsweise betriebswichtigen - Zustand überhaupt erreicht. Im Gegensatz zum Testen überprüft der Model-Checker dabei den gesamten Zustandsraum des Systems und somit das gesamte Verhalten auf Konformität mit der Spezifikation. Mit Model-Checking kann daher ein Korrektheitsnachweis auf dem Niveau eines mathematischen Beweises durchgeführt werden [CGP00, S. 1].

Grundsätzlich gibt es jedoch deutliche Einschränkungen bezüglich der Komplexität des zu prüfenden Systems, da Modellchecker systembedingt dem Problem der so genannten Zustandsraumexplosion unterworfen sind. Dieses Phänomen bezeichnet das rasche Anwachsen des Speicherbedarfs, insbesondere bei Systemen mit nebenläufigen Prozessen, das bereits bei mäßig komplexen Systemen zur Nicht-Anwendbarkeit des Model-Checking führen kann [ARA05, S. 152 ff.].

Das Ergebnis des Model-Checkings ist entweder die Aussage, dass das Verhaltensmodell die Spezifikation nicht verletzt, oder aber ein Gegenbeispiel, das diejenige Folge von Stimuli zeigt, die zu einer Verletzung der Randbedingungen führen. Arabestani [ARA05] zeigte in seiner Arbeit, wie sich diese Technologie prinzipiell auf ingenieurwissenschaftliche Problemstellungen anwenden lässt.

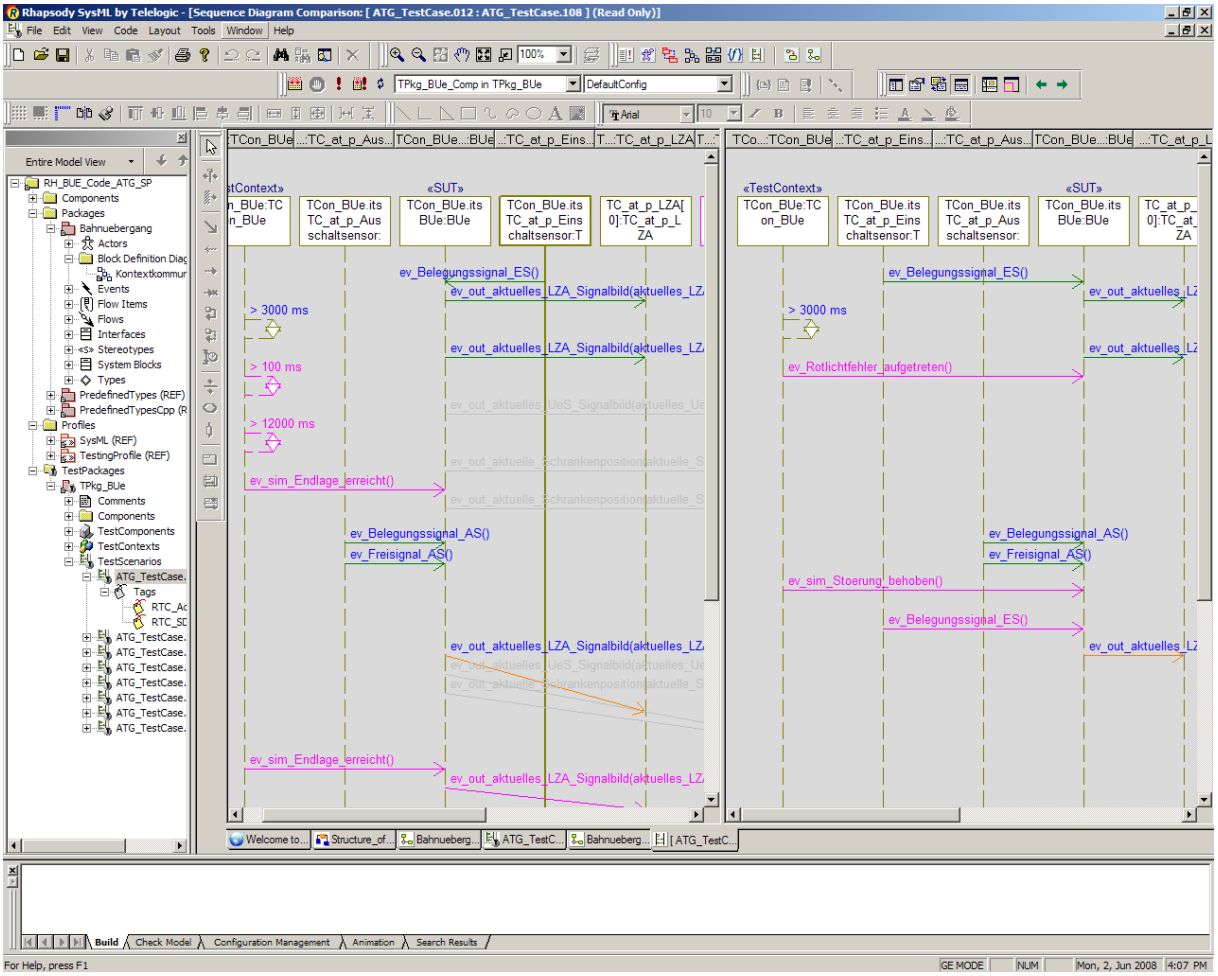


Abbildung 6.24.: Automatisierter Vergleich von Sequenzdiagrammen

Auch im Rahmen von OPRAIL wurde der Model-Checker RUVE [STM04] für die Überprüfung von UML-Modellen erprobt. Allerdings war dies auf Grund des eher experimentellen Charakters der Software nur für sehr einfache und ausgewählte Modelle möglich. Auch zum jetzigen Zeitpunkt existiert kein für den Produktiveinsatz taugliches Werkzeug zur Durchführung des Model-Checking von UML/SysML-Modellen. Zudem ist das Modelchecking zurzeit auf die Verifikation von Verhaltensmodellen aus Zustandsmaschinen begrenzt. Dies steht im Widerspruch zu der in dieser Arbeit propagierten Verwendung von Aktivitätsdiagrammen für die Verhaltensmodellierung. Zwar zeigt Eshuis in [ESH06] einen Ansatz, mit dem auch Aktivitätsdiagramme formal verifiziert werden können. Allerdings fehlt dafür jedoch noch eine entsprechende technische Umsetzung in ein praxisnah einsetzbares Werkzeug.

Daher wird die formale Verifikation im Rahmen dieser Arbeit auch nur als mögliche Zukunftsoption zur formalen Überprüfung von Anforderungsmodellen aufgeführt. Somit erfolgt auch keine exakte Definition des Subprozesses, da diese vom tatsächlich verwendeten Model-Checker abhängig wäre. Davon unabhängig sind jedoch die Eingangsdaten, die für die formale Verifikation erforderlich sind: die Verhaltenssicht des Anforderungsmodells und die zu überprüfenden Sicherheits- und Lebendigkeitsbedingungen. Letztere werden üblicherweise durch temporallogische Formeln beschrieben. Jedoch existieren auch Ansätze, Sequenzdiagramme so umzuformen, dass sie für diesen Zweck verwendet werden können [ARA05, S. 151]. Auf diese Weise ließe sich die für Fachingenieure unbeliebte Auseinandersetzung mit formalen Beschreibungsmitteln

vermeiden. Der Model-Checker verwendet diese Eingangsdaten und führt damit die formale Verifikation durch. Stellt er dabei eine Verletzung der Randbedingungen fest, muss der Subprozess abgebrochen und - ähnlich zum Subprozess S5 - in der übergeordneten Phase P2 der zur Verletzung der Randbedingungen führende Modellierungsfehler ermittelt werden.

6.4.7. S7 - Subsystemsicht bearbeiten

Der Subprozess S7 umfasst alle Aktivitäten, die zur Modellierung der Subsystemsicht erforderlich sind. Pro Ausführung des Subprozesses wird eine Subsystemkomponente spezifiziert und in die Subsystemarchitektur eingeführt. Dazu sind im Subprozess S7 zwei Gruppen von Aktivitäten definiert. Die erste Gruppe umfasst Tätigkeiten, die direkt die Modellierung der Subsystemkomponenten und der Subsystemarchitketur definieren. Die zweite Gruppe umfasst gegebenenfalls erforderliche Anpassungstätigkeiten in anderen Inhaltselementen der Subsystemsicht. Zur ersten Gruppe gehören:

- Erstellung/Überarbeitung der Subsystemkomponenten (auf Blockebene, die konkrete Verwendung der Komponente erfolgt in der Aktivitität „Subsystemverwendung erstellen/-überarbeiten")
- Subsystemverwendung erstellen/überarbeiten (integriert die einzelnen Subsystemkomponenten in der Subsystemarchitektur)
- Konnektoren erstellen/überarbeiten (diese Aktivitäten verbinden die einzelnen Subsystemkomponenten miteinander)

Zur zweiten Gruppe gehören:

- Portdefinitionen erstellen/überarbeiten
- Signale erstellen/überarbeiten
- Informationsflüsse erstellen/überarbeiten

Bild 6.25 zeigt das Aktivitätsdiagramm für den Subprozess S7 mit den oben aufgeführten Aktivitäten.

6.4.8. S8 - nicht-funktionale Anforderungen überarbeiten

In diesem Subprozess sind alle Aktivitäten gebündelt, die die nicht-funktionalen Anteile der Anforderungsspezifikation betreffen. Ziel dieses Subprozesses ist die Anpassung der nicht-funktionalen Anforderungen an den Fortschritt der funktionalen Modellierung und die Aktualisierung der Beziehungen zwischen beiden Teilen der Anforderungsspezifikation. Dieser wird in allen Phasen des Prozesses aufgerufen, damit die Auswirkung von Änderungen an einem der beiden Bestandteile des Anforderungsmodells möglichst schnell berücksichtigt werden können. Dazu sind innerhalb des Subprozesses die folgenden drei Aktivitäten definiert:

- Nicht-funktionale Anforderungen erstellen/überarbeiten - hierbei werden neue relevante nicht-funktionale Anforderungen erfasst und ggf. bestehende Anforderungen überarbeitet. Dies geschieht gemäß dem in Abschnitt 5.2.3 beschriebenen Konzept zwar außerhalb des funktionalen Modells, die erfassten natürlich-sprachlichen Informationen werden allerdings automatisiert als SysML-Requirements in dieses übernommen.
- Textfragmente aus externen Dokumenten erstellen/überarbeiten - hierbei handelt es prinzipiell um das gleiche Vorgehen wie im vorherigen Arbeitsschritt, jedoch bezieht sich diese

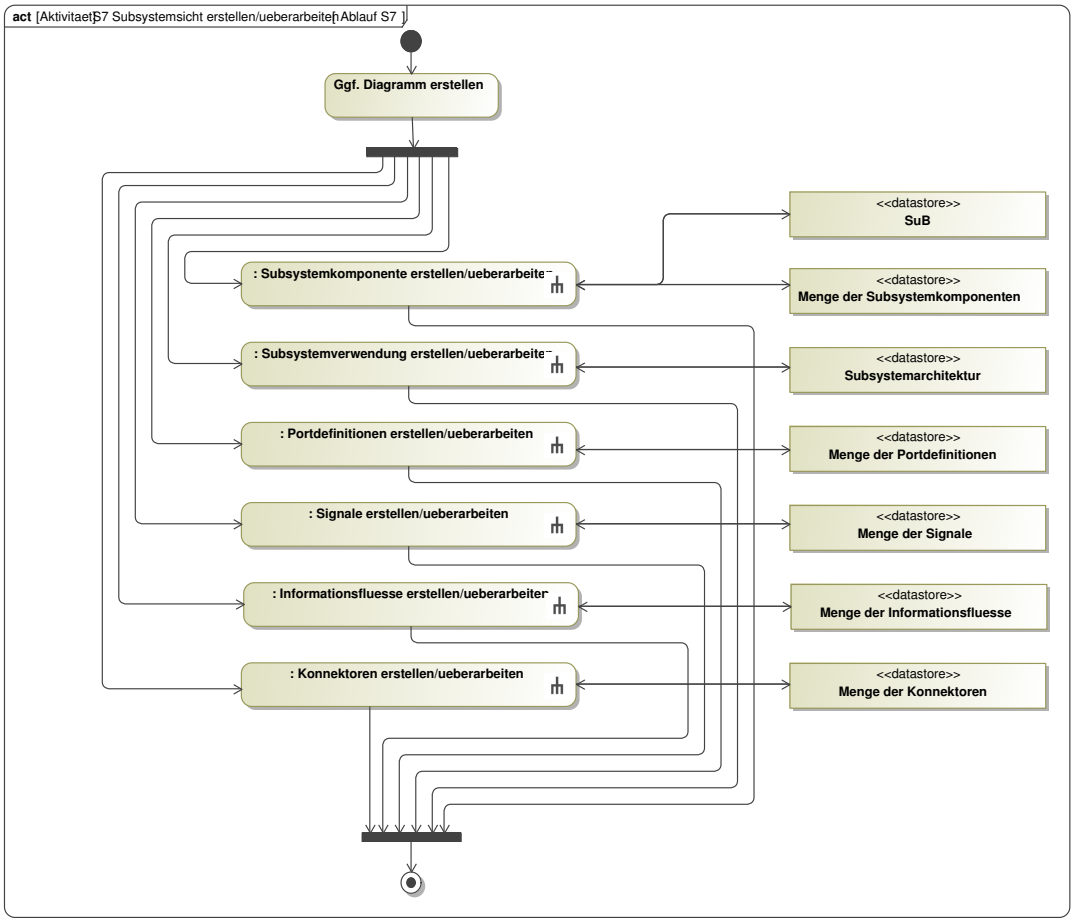


Abbildung 6.25.: Aktivitätsdiagramm Subprozess S7

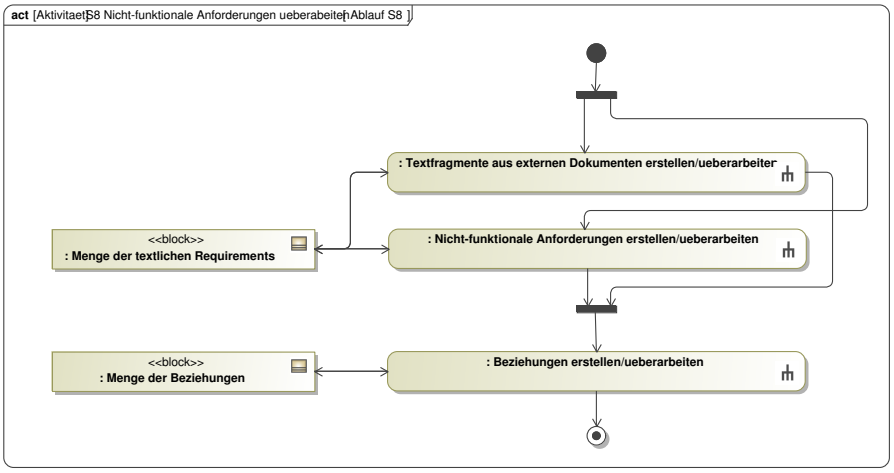


Abbildung 6.26.: Aktivitätsdiagramm Subprozess S8

Aktivität nicht auf vom Modellierer erstellte Anforderungen, sondern um unveränderliche Passagen externer Dokumente, Vorgaben und Verordnungen.

- Beziehungen erstellen/überarbeiten - in dieser Aktivität wird geprüft, ob sich durch den Fortschritt der Modellierung neue Beziehungen zwischen Elementen des funktionalen Modells und den nicht-funktionalen Anforderungen ergeben haben, neue beispielsweise `<<satisfy>>`- oder `<<verify>>`-Beziehungen. Wenn ja müssen sie in dieser Aktivität ergänzt werden.

Abbildung 6.26 zeigt die beschriebenen Abläufe dieses Subprozesses als Aktivitätsdiagramm.

6.5. Dokumentation des funktionalen Modells

Nach Anwendung des in den vorherigen Kapiteln beschriebenen Prozesses liegt das funktionale Anforderungsmodell inklusive Verknüpfungen zu externen Dokumenten und zu nicht-funktionalen Anforderungen als Datei oder Datenbank des gewählten Modellierungswerkzeuges vor.

Auch wenn zu diesem Zeitpunkt die inhaltliche Arbeit am Modell - vorbehaltlich dessen Fortschreibung - abgeschlossen ist, wird sich in vielen Anwendungsfällen als weiterer Arbeitsschritt die Dokumentation des Modellinhalts anschließen. Diese ist üblicherweise aus den folgenden Gründen sinnvoll und erforderlich:

(a) Wie viele andere Software auch, entwickeln sich UML- und SysML-Werkzeuge in kurzen Zyklen weiter. Dies kann dazu führen, dass bereits nach einigen Jahren ein Modell in einem bestimmten Datenformat nicht mehr gelesen werden kann. Erschwerend kommt hinzu, dass im Bereich sicherheitskritischer Eisenbahnsysteme sehr lange Dokumentationsfristen üblich sind. Auch der XMI-Standard [OMG07-4], der als Austauschformat für UML-Modelle entwickelt wurde, löst auf Grund mangelnder Unterstützung durch die Werkzeughersteller dieses Problem nur unzureichend.

(b) Der direkte Austausch von Modellen als Arbeitsgrundlage zwischen den Projektbeteiligten setzt voraus, dass alle Partner die gleiche oder zumindest eine kompatible Werkzeugkette verwenden. Davon ist auf Grund der Vielzahl an UML/SysML-Werkzeugen jedoch

nicht auszugehen. Die zusätzliche, projektspezifische Anschaffung eines weiteren Werkzeugs scheitert üblicherweise an den sehr hohen Kosten sowohl für die Software an sich, als auch für die dann zusätzlich erforderliche Ausbildung von Mitarbeitern.

(c) Ein nach dem in dieser Arbeit beschriebenen Vorgehen erstelltes Anforderungsmodell kann mehr Informationen enthalten, als für eine konkrete Spezifikationsaufgabe benötigt wird. Dies ist beispielsweise dann der Fall, wenn sich das Modell auf ein umfangreiches System bezieht, das (entsprechend dem Ebenenkonzept) aus mehreren Subsystemen zusammengesetzt ist. Bei der Ausschreibung eines Teilsystems wäre es dann nicht sinnvoll, alle Modellbestandteile weiterzugeben. Vielmehr muss eine qualifizierte Auswahl der für eine konkrete Anforderungsspezifikation erforderlichen Diagramme erfolgen.

Um diese Problembereiche zu umgehen, soll im Folgenden ein Dokumentationskonzept beschrieben werden. Mit diesem lässt sich als Endprodukt des Prozesses eine dauerhafte, mit gängigen Office-Programmen lesbare und ausdruckbare Dokumentation aller relevanten Inhalts des funktionalen Modells erstellen.

Grundlegender Gedanke ist dabei, dass funktionale Modell als Arbeitsmittel für Analyse und Erstellung, nicht aber für die dauerhafte Archivierung aufzufassen. Die intensiven Arbeitsphasen am Modell (also im Wesentlichen die Phasen P0 - P3) finden meist in einem zeitlichen Rahmen statt, in dem eine durchgängige Werkzeugverfügbarkeit sichergestellt ist. Diese Phasen können und sollen daher weitgehend innerhalb des Modellierungswerkzeugs durchgeführt werden. Für die anschließende Dokumentation muss hingegen ein Weg gefunden werden, die Arbeitsergebnisse in eine archivierbare Form zu übertragen.

Dazu bietet es sich an, für eine spezifische Aufgabe - z.B. für ein konkretes Lastenheft - ein Dokument in einem Anforderungsmanagement-Werkzeug wie IBM/telelogic DOORS oder einer Office-Textverarbeitung zu erstellen. Der Aufbau dieses Dokumentes kann sich dabei frei von den Gliederungsstrukturen im Modell an den üblichen Vorgaben der jeweiligen Organisationseinheit orientieren. An bestimmten Positionen innerhalb dieses Dokuments werden dann inhaltlich passende Diagramme aus dem Modell eingefügt. Dadurch ergibt sich ein Spezifikationsdokument, dessen äußere Struktur ähnlich zu bestehenden, textbasierten Dokumentationen ist. Gleichzeitig enthält es die relevanten - und nur die relevanten - Elemente aus dem Modell. Der Hub an Spezifikationsqualität gegenüber klassischer textbasierter Spezifikation wird dabei dadurch erzielt, dass die in den Diagrammen dargestellten Sachverhalte auf den konsistenten, verifizierten und validierten Inhalten des funktionalen Modells basieren. Zusätzlich lassen sich so einzelne Elemente des Modells durch eine eindeutige Kennziffer identifizieren[5]. Damit erhält man eine eindeutige Referenz, mit der sich die Diagramme aus dem Modell, z.B. bei Reviews und Prüfungen, gezielt ansprechen lassen.

Im Spezifikationsdokument werden üblicherweise neben den Diagrammen aus dem Modell noch ergänzende Texte erforderlich sein. Diese können im ungünstigen Fall Widersprüche zu den im den Diagrammen dargestellten Sachverhalte enthalten. Um diese Widersprüche auszuschließen bietet es sich an, die Verbindlichkeit einzelner Bestandteile des Spezifikationsdokuments durch zusätzliche Attribute zu steuern. Sinnvoll wäre beispielsweise, alle Diagramme mit einer „muss"-Verbindlichkeit auszuzeichnen und alle erklärenden Texte mit einem „Info"-Attribut. Dadurch würde festgelegt, dass nur die aus dem funktionalen Modell stammenden Diagramme verbindlicher Spezifikationsbestandteile und in Zweifelsfällen Aussagen in den ergänzenden textlichen Passagen vorzuziehen sind. Die Abbildung 6.27 verdeutlicht dieses Konzept.

[5] DOORS vergibt beispielsweise für jede neue Zeile in einem Spezifikations-Modul eine lebenslang eindeutige Kennzahl, die im gleichen Modul auch nach Löschung niemals neu vergeben wird.

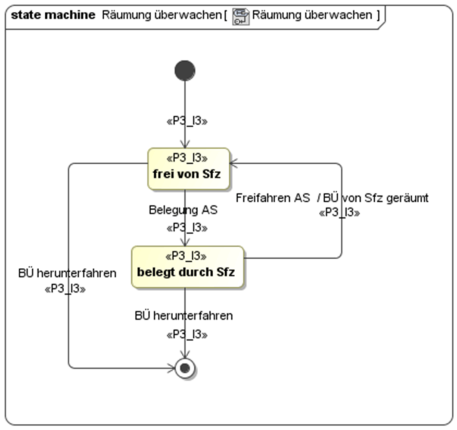
ID	Verbindlichkeit	Inhalt
…	…	…
TAS.104	Muss	
TAS.156	Info	Das obige Diagramm zeigt die Zustandsmaschine, die das Verhalten des BÜ bei der Räumung beschreibt.
TAS.99	Info	Die dadurch entstehenden Kommunikationen zwischen BÜ und Schienfahrzeug sind im folgenden Sequenzdiagramm dargestellt
…	…	…

Abbildung 6.27.: Beispielhafte Ansicht eines Spezifikationsdokuments mit eingebettetem Diagramm

Die nach diesem Vorgehen erstellten Spezifikationsdokumente lassen sich problemlos in archivierbare Formate (wie z.B. PDF/A [ISO19005]) exportieren und ausdrucken. Damit lässt sich sowohl die Dokumentationsaufgabe erfüllen, als auch die Informationsmenge in einem Spezifikationsdokument sehr gut steuern. Exporte solcher Spezifikationen in Formate von Office-Paketen erlauben beispielsweise Reviews durch Partner, die nicht über eigene Installationen der Modellierungswerkzeuge verfügen.

7. Zusammenfassung, Ergebnisse und Ausblick

In der vorliegenden Arbeit wurde ein auf dem BMW-Prinzip basierendes Konzept zur Modellierung von Anforderungen vorgestellt. Dazu wurde zunächst die aktuelle Situation sowohl bei der Anforderungserstellung, als auch bei der sich anschließenden Phase der Systementwicklung untersucht. Dabei zeigte sich, dass bei der Systementwicklung die Anwendung von semi-formalen, objektorientierten Beschreibungsmitteln und daran angepassten Vorgehensmodellen viel weiter fortgeschritten ist, als im Bereich der Anforderungsmodellierung. Dort haben sich Konzepte zur Modellierung funktionaler Anforderungen und entsprechende Prozesse zur ihrer Erstellung noch nicht durchsetzen können. Gleichzeitig ist die Anforderungserstellung aber diejenige Phase, in der die meisten Fehler in das spätere System eingebracht werden. Daraus wurde die Notwendigkeit abgeleitet, das aktuelle Vorgehen bei der Anforderungsspezifikation für Bahnsysteme zu optimieren.

Das im Rahmen dieser Arbeit entwickelte Konzept basiert dabei auf zwei wesentlichen Bestandteilen: Der Entwicklung einer praxistauglichen Struktur für eine Anforderungsspezifikation auf Basis der Modellierungssprache SysML(A) einerseits und andererseits die Ableitung eines geeigneten Prozessmodells, das die Erstellung der Anforderungsspezifikation steuert und die einzelnen Arbeitsschritte spezifiziert.

Kernelement der Anforderungsspezifikation ist das funktionale Modell. Zur Ermittlung eines geeigneten Beschreibungsmittels für dieses Modell wurden verschiedene prinzipielle Möglichkeiten analysiert, unter anderem natürliche und formale Sprachen, sowie semi-formale, objektorientierte Beschreibungsmittel. Letztendlich wurde ein Subset der Systems Modeling Language (SysML) als geeignet identifiziert, das als SysML(A) bezeichnet wird und alle für die Modellierung von Anforderungen notwendigen Konstrukte enthält. Der Sprachumfang von SysML(A) wurde in einem Top-Down-Vorgehen festgelegt. Es analysiert die einzelnen Aspekte, die innerhalb des Modells berücksichtigt werden müssen und verfeinert deren Abbildung bis auf die Ebene einzelner Sprachelemente. Im funktionalen Anforderungsmodell werden diese Aspekte durch einzelne Sichten repräsentiert, die spezifische Informationen innerhalb des Modells bündeln. Als weitere Gliederungsdimension wurden einzelne Ebenen eingeführt, mit denen sich durch eine einfache, zweistufige Hierarchie von Systemen und Subsystemen alle denkbaren Systemarchitekturen beschreiben lassen.

Ein wesentliches Ziel beim Entwurf des funktionalen Anforderungsmodells war dessen möglichst frühe Ausführbarkeit. Zur Erreichung dieses Ziels wurden zunächst die prinzipiellen Anforderungen für die Ausführbarkeit von Modellen untersucht und anschließend eine primär auf Aktivitätsdiagrammen beruhende Modellierungsstrategie des Systemverhaltens vorgeschlagen. Diese ermöglicht - trotz noch offener Diskussionspunkte bezüglich der Formalität von Aktivitätsdiagrammen - eine für den Modellierer einfach durchzuführende Verhaltensspezifikation und die Modellausführung mit entsprechenden Werkzeugen. Dadurch werden bereits bei der Erstellung der Systemanforderungen Technologien wie Test des Modells, Testfallgenerierung und formale Verifikation zugänglich. Somit lässt sich die Qualität der Anforderungsspezifikation gegenüber einer rein textlichen Beschreibung deutlich steigern.

Die hier vorliegende Arbeit greift damit Ideen aus älteren Arbeiten in diesem Bereich (wie beispielsweise die Dissertation von Arabestani [ARA05]) auf. Sie passt diese an die aktuelle Situation an und ergänzt sie um Elemente, die für eine Migration hin zur modellbasierten Anforderungsspezifikation im Produktiveinsatz zwingend erforderlich sind. Zwei wichtige Elemente sind dabei der Übergang zur gegenüber der UML besser geeigneten SysML(A) als Modellierungssprache sowie die beschriebenen Mechanismen für die Modellorganisation. Für diese beiden Aspekte wurden geeignete Lösungen im Rahmen dieser Arbeit aufgezeigt. Über den erreichten Arbeitsstand hinaus lassen sich aber auch weitere, zukünftige Entwicklungen skizzieren: So wäre es beispielsweise sinnvoll, wenn auch für Aktivitätsdiagramme eine vollständige, formale Semantikdefinition entwickelt werden würde. Zusammen mit den formalen Definitionen der UML-Modellelemente in [ARA05] - die an die Eigenschaften der SysML angepasst werden müssten - läge damit für alle in der SysML(A) verwendeten Sprachelemente eine formale Definition vor. Dies würde es ermöglichen, die Modelle der Anforderungsspezifikation direkt für die Systementwicklung weiterzuverwenden und sie durch Transformationen, Verfeinerung und Ergänzungen so zu modifizieren, dass sie unmittelbar zur Erzeugung des Produktivcodes für die Zielplattform eingesetzt werden können. Damit könnte eine mit SCADE vergleichbare Modell-Durchgängigkeit für den gesamten Lebenszyklus erreicht werden. Im Gegensatz zu SCADE lägen die Modelle jedoch in einer offenen, standardisierten Beschreibungssprache vor.

Weiteres interessantes Forschungsgebiet sind Metamethoden für die Beschreibung des Systemverhaltens. Wie in Abschnitt 6.4.4 beschrieben, wird von Hon momentan an der Umsetzung von Entscheidungstabellen in aussagenlogische Formeln geforscht. Diese könnten dann in Verhaltensdiagrammen genutzt werden und deren Erstellung vereinfachen. Damit wäre es einem Modellierer von Anforderungsmodellen möglich, sich stärker auf die fachlichen Inhalte des Modells zu konzentrieren, als auf deren Umsetzung durch bestimmte Sprachelemente von Aktivitäts- und Zustandsdiagrammen. Es erscheint lohnenswert, diesen Ansatz einer zusätzlichen Abstraktionsebene oberhalb der in der SysML oder UML definierten Verhaltensdiagramme weiterzuverfolgen. So könnten in Zukunft neben den für logische Aufgabenstellungen geeigneten Entscheidungstabellen ähnliche Konzepte auch für andersartige Problemstellungen integriert werden.

Weiterhin wird auch die Integration nicht-funktionaler Anforderungen berücksichtigt. Das für diese Art von Anforderungen gewählte Konzept geht davon aus, dass sich nicht-funktionale Anforderungen üblicherweise inhaltlich schlecht durch Konstrukte der SysML(A) ausdrücken lassen. Vielmehr erwies es sich als zielführend, nicht-funktionale Anforderungen als freitextliche Elemente abzubilden, die über definierte Relationen an Entitäten des funktionalen Modells angekoppelt werden. Das in der vorliegenden Arbeit beschriebene Konzept basiert auf der externen Verwaltung der nicht-funktionalen Anforderungen. Diese werden in das funktionale Modell gespiegelt und mit Modellartefakten durch `<<trace>>`-, `<<verify>>`-, `<<satisfy>>`-Verknüpfungen sowie über einfache Dependency- und Containment-Relationen verbunden. Das beschriebene Vorgehen kann zudem dazu verwendet werden, um die Erfüllung externer, unveränderlicher Vorgaben, Normen und Richtlinien durch das Anforderungsmodell zu dokumentieren. Genauso lassen sich über diesen Mechanismus auch informelle Anforderungen aus der Frühphase der Systementwicklung an das Modell ankoppeln.

Als zweites Kernelement der vorliegenden Arbeit wurde ein Prozess entwickelt, der die Erstellung der Anforderungsspezifikation von ersten Ideen bis hin zur Lebenszyklusbegleitung eines fertigen Systems strukturiert und steuert. Dieses Prozessmodell füllt somit die in der Analyse erkannte methodische Lücke im BMW-Konzept aus. Zur Ableitung eines geeigneten Prozessmodells wurden zunächst die besonderen Eigenschaften der Erstellung von Anforderungsspezifikationen ermittelt. Anschließend wurden verschiedene Arten von Vorgehensmodellen untersucht, die als Basis für das zu definierende Prozessmodell dienen. Letztendlich wur-

de ein iterativ-inkrementelles Vorgehensmodell ausgewählt, das die Ideen aus dem OPRAIL-Forschungsvorhaben [OPR06] aufgreift und an die besonderen Belange der Anforderungsmodellierung anpasst. Dies betrifft vor allem die Problematik der nur allmählich anwachsenden Informationsmenge, weswegen die ursprünglich für die Systementwicklung gedachten Vorgehensmodelle nicht direkt verwendet werden können. Zur Anpassung wurde daher das Konzept der Aufgabenorientierung eingeführt, das auf dem zeitlich gestaffelten Beginn einzelner Modellierungsaufgaben beruht. Zur Definition des Prozesses wurden Phasen, Subprozesse, Iterationen und Aktivitäten definiert und in einem Metamodell zu einer Prozessdefinition kombiniert. Die in den einzelnen Phasen und Subprozessen durchzuführenden Arbeitsschritte wurden durch Aktivitätsdiagramme beschrieben. Diese Beschreibung erfolgte derart, dass genügend Freiraum für die Anpassung des Prozesses an verschiedene Einsatzszenarien verbleibt, dennoch aber die generelle Zielrichtung der prozessgetriebenen Anforderungserstellung deutlich wird.

Ebenfalls in die Prozessbeschreibung integriert sind Vorgaben zur Anwendung von Test- und Verifikationstechniken. Letztere lassen sich auf Grund des momentanen Entwicklungsstands der entsprechenden Softwaretools und dem Fehlen einer vollständigen Semantik-Definition für SysML(A) momentan noch nicht im Produktivumfeld einsetzen. Die Prozessdefinition ist allerdings auf deren Einsatz vorbereitet. Bereits gezeigt werden konnte im Rahmen dieser Arbeit hingegen die Anwendung von Testausführungs- und Testfallgenerierungswerkzeugen. Sie werden im Prozess zur Erfüllung von drei wesentlichen Zielen verwendet: Zur Sicherstellung der Konsistenz zwischen Szenarien- und Verhaltenssicht, zur Prüfung des Systemverhaltens nach jedem Iterationsschritt der Verhaltensmodellierung und zur Ermittlung neuer Szenarien, um implizit modelliertes Systemverhalten explizit darzustellen. Die Auswahl der Methoden und Werkzeuge für die Testfallgenerierung erfolgte in einem mehrstufigen Verfahren, in dem zunächst Anforderungen aufgestellt und dann geeignete Werkzeuge anhand einer Taxonomie identifiziert wurden.

Abschließend wird in dieser Arbeit eine Methodik vorgestellt, mit der es möglich ist, die Modellierungsergebnisse langfristig zu sichern und zu archivieren. Dazu werden die relevanten Diagramme aus dem Modell in ein Anforderungsmanagement-Werkzeug übertragen und dort in ein Spezifikationsrahmenwerk eingefügt. Aus diesem Werkzeug heraus kann die Anforderungsspezifikation dann in Datenaustauschformate wie PDF-A exportiert oder ausgedruckt werden.

Daraus ergibt sich eine präzise Unterteilung für den gesamten Lebenszyklus der Anforderungsspezifikation:

- Die Analyse- und Erstellungs-Phase profitiert von den Vorteilen des modellbasierten Ansatzes gegenüber rein textlicher Spezifikation. Obwohl ein Modell nicht wie eine Theorie notwendigerweise vollständig und korrekt ist, stellt eine modellhafte Beschreibung eines Systems eine gute Basis für Konsistenz- und Adäquatheitsüberprüfungen dar, die eine deutliche Verbesserung des semantischen Inhalts der Anforderungsspezifikation ermöglicht.
- In der Dokumentations- und Archivierungsphase stellt die Integration der Modellbestandteile in einen strukturierten, textbasierten Spezifikationsrahmen die langfristige Nachvollziehbarkeit und Verfügbarkeit der Anforderungsspezifikation sicher. Gleichzeitig wird damit das Endprodukt der Spezifikationsarbeit vom ursprünglich verwendeten Werkzeug unabhängig.

Insgesamt wird somit erstmalig eine Prozessdefinition für die Erstellung von Anforderungsspezifikationen sicherheitskritischer Systeme im Bahnbereich vorgestellt. Damit wird die Anwendung der bisher nur im Bereich der Systementwicklung etablierten Prozessmodelle auf einen zeitlich vorgelagerten Bereich des Systemlebenszyklus ausgedehnt. Damit wird auch dort die im BMW-Prinzip geforderte Kombination von Beschreibungsmitteln, Methoden und Werkzeugen umgesetzt.

Durch die in dieser Arbeit erreichten Ziele wird ein Ersteller von Anforderungsspezifikationen in die Lage versetzt, mit dem definierten Subset SysML(A) die funktionalen Anforderungen von zukünftigen Systemen zu beschreiben und das Modell so zu strukturieren, dass die darin enthaltenen Informationen leicht zugänglich sind. Er kann das modellierte Verhalten gegen seine eigenen Verhaltensvorgaben prüfen, und - sobald die entsprechenden Voraussetzungen gegeben sind - gegen Sicherheitsrandbedingungen formal verifizieren. Der Modellierer wird bei der Anforderungserstellung durch eine Prozessvorgabe geführt, die sicherstellt, dass alle notwendigen Aspekte des Systems berücksichtigt werden und zudem eine sinnvolle Reihenfolge bei der Modellerstellung eingehalten wird. Insgesamt kann damit ein Qualitätsniveau für Anforderungsspezifikationen erzielt werden, das weit über demjenigen rein natürlich-sprachlicher Dokumente liegt.

Literaturverzeichnis

[ALE02] ALEXANDER, Ian F.; STEVENS, Richard: *Writing Better Requirements,* 1. Aufl., Addison-Wesley Professional, 2002 - ISBN: 0321131630

[ARA05] ARABESTANI, Saeid: *Formal verifizierbare objektorientierte Systemspezifikationen mit UML für Eisenbahnsicherungssysteme*, Technische Universität Braunschweig, FB 6: Bauingenieur- und Vermessungswesen, Diss. 2005. - ISBN 3923325673

[ARG00] ARABESTANI, Saeid; GAYEN, Jan-Tecker: *Objektorientierte Analyse zur Modellierung im Eisenbahnwesen.* In: Signal+Draht (1+2) 2000; S. 20 ff.

[BDM98] BEHM, Patrick; DESFORGES, Pierre; MEYNADIER, Jean-Marc: *METEOR: An Industrial Success in Formal Development*. In: Second conference on the B method, Montpellier, France, 1998

[BOE88] BOEHM, Barry W: *A Spiral Model of Software Development and Enhancement.* In: IEEE Computer. Vol. 21, Ausg. 5, Mai 1988, S. 61-72

[BRA05] BRABAND, Jens: *Risikoanalysen in der Eisenbahnautomatisierung*, 1. Aufl. Hamburg: Eurailpress Tetzlaff-Hestra GmbH & Co. KG, 2005. - ISBN 3777103357

[BCD07] BROY, Manfred; CRANE, Michelle; DINGEL, Jürgen; HARTMAN, Alan; RUMPE, Bernhard; SELIC, Bran: *2nd UML 2 Semantics Symposium: Formal Semantics for UML.* In: Models in Software Engineering. Workshops and Symposia at Models 2006. Genoa. LNCS 4364, Springer, Januar 2007

[CGP00] CLARKE, Edmund M.; GRUMBERG, Orna; PELED, Doron: *Model Checking*, 2. Aufl. Cambridge: The MIT Press, 2000. - ISBN 0262032708

[COC03] COCKBURN, Alistair: *Use-Cases effektiv erstellen,* 1. Aufl. Bonn: mitp-Verlag, 2003 - ISBN: 3826613449

[COC01] COCKBURN, Alistair: *Agile Software Developement*, 1. Aufl., Addison-Wesley Professional, 2001 - ISBN: 0201699699

[COC93] COCKBURN, Alistair: *Unraveling incremental developement,* http://alistair.cockburn.us/index.php/Unraveling_incremental_development, Abruf vom 01.06.2008

[COH08] COHEN, Moshe; Telelogic AB (Hrsg.): *Model Driven Testing: Your Secret to Survival*; Telelogic White Paper; Version 2.1, 2008

[CPH87] CASPI, P.; PILAUD, D.; HALBWACHS, N.; PLACIE, J. A.: *LUSTRE: a declarative language for real-time programming.* In: Annual Symposium on Principles of Programming Languages, Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, S. 178 - 188, München, 1987

[DB09] Deutsche Bahn AG, Systemverbund Bahn, DB Systemtechnik (Hrsg.): *Schnittstelle zwischen Weichenanschaltbaugruppe und Weichenantrieb - funktionale Anforderungsspezifikation (FAS)* -, Version 1.0, 2009.

[DOD95] US DEPARTEMENT OF DEFENSE (Hrsg.): *Parametric Cost Estimating Handbook*, 1995

[DOU01] DOUGLASS, Bruce Powell: *UML for executable specification*. In: EDN Magazine 16.08.2001; S. 83 ff.

[DUB06] DUBROVIN, Jori; Helsinki University of Technology, Laboratory for Theoretical Computer Science, Research Report A101: *JUMBALA - an action language for UML state machines*; 2006; Dokument: HUT-TCS-A 101

[EN50121] Norm DIN EN 50121:2006. *Bahnanwendungen - Elektromagnetische Verträglichkeit*

[EN50125] Norm DIN EN 50125:2000. *Bahnanwendungen - Umweltbedingungen für Betriebsmittel*

[EN50126] Norm DIN EN 50126:2000. *Spezifikation und Nachweis der Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS)*

[EN50129] Norm DIN EN 50129:2003. *Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme – Sicherheitsrelevante elektronische Systeme für Signaltechnik*

[ESH06] ESHUIS, Rik; Eindhoven University of Technology: *Symbolic model checking of UML activity diagrams; 2006.*

[ESW01] ESHUIS, Rik; WIERINGA, Roel; University of Twente, Departement of Computer Science: *A Formal Semantics for UML Activity Diagrams - Formalising Workflow Models*; 2001.

[FUS96] FUCHS, E.; SCHWITTER, R.: *Attempto Controlled English (ACE)* (CLAW 96, First International Workshop on Controlled Language Applications), University of Leuven, Belgium, 1996

[GAP94] GAYEN, Jan-Tecker; PASTERNOK, Thomas: *Bewertungskriterien für Methoden zur Erstellung einer Anforderungsspezifikation*; SOSAT 3 Forschungsbericht; Braunschweig: Institut für Eisenbahnwesen und Verkehrssicherung, 1994

[GEA08] GEORGE, Gerhard; AST, Dietmar: *Rollenverteilung zwischen Auftraggeber und Auftragnehmer im RAMS-Prozess.* In: (eb) Elektrische Bahnen 6 (2008); S. 278 ff.

[HAR87] HAREL, David: *Statecharts: A visual Formalism for Complex Systems*. In: Sciences of Computer Programming 8 (1987), S. 231 ff.

[HAP02] HAXTHAUSEN, Anne; PELESKA, Jan: *A domain specific language for railway control systems.* In: Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology, IDPT2002, Pasadena, California, 2002

[HGE08] HON, Yuen Man; GAYEN, Jan-Tecker; EHRICH, Hans-Dieter: *OOLH: A formal framework for specifying system requirements*; 2008

[HOF08] HOFFMANN, Hans-Peter; Telelogic AB (Hrsg.): *SysML-Based Systems Engineering Using a Model-Driven Developement Approach*; Telelogic Whitepaper; Version 1; 2008

[IEEE830] Norm IEEE Std 830-1993. *Recommended Practice for Software Requirements Specifications*, 1993

[IEC60812] Norm IEC 60812:2001. *Analysetechniken für die Funktionsfähigkeit von Systemen - Verfahren für die Fehlerzustandsart- und -auswirkungsanalyse (FMEA)*

[IEC13568] Norm IEC 13568:202. *Information technology — Z formal specification notation — Syntax, type system and semantics*

[ISO19005] Norm ISO 19005-1. *Document Management - Electronic document file format for long term preservation - Part 1: Use of PDF 1.4 (PDF/A-1)*, 2005

[ITU04] International Telecommunication Union (Hrsg.): *ITU-T Recommendation Z.120 - Message Sequence Chart (MSC),* Genf: ITU, 2004.

[JZM07] JIANG, Ke; ZHANG, Lei; MIYAKE, Shigeru: *OCL4X: An Action Semantics Language for UML Model Execution* (COMPSAC 2007 - Computer Software and Applications Conference). 31st Annual International Volume 1, 2007.

[KBS06] Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (Hrsg.): *V-Modell XT,* Version 1.0.2/2.0.5/1.2.1, 2006

[KM07] KURZ, Sonja-Lara; MILIUS, Birgit: *Der Functional-Dependency-Test zur Ableitung von Teilfunktionen*. In: Signal+Draht 11 (2007); S. 28 ff.

[KNO07] KNOLLMANN, Volker: *UML-basierte Testfall- und Systemmodelle für die Eisenbahn Leit- und Sicherungstechnik,* Technische Universität Braunschweig, Fakultät für Maschinenbau, Diss. 2007

[KOR815] Konzernrichtlinie 815; Hrsg: DB Netz AG: *Bahnübergänge planen und instandhalten*, 2002

[LAM02] LAMPORT, Leslie: *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Professional, 2002. - ISBN: 032114306X

[LET05] LETTRARI, Marc: *Efficient state space exploration of reactive object-oriented programs*, Universität Oldenburg, Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften, Department für Informatik, Diss. 2005

[LOR97] LORENZ, Manfred: *Stellwerkstechnik, Band 1: Bahnhofssicherungstechnik,* Selbstverlag, 1997

[MAR79] DEMARCO, Tom: *Structured Analysis and System Specification.* Prentice Hall International, 1979. - ISBN: 0138543801

[MEY75] MEYHAK, Hermann: *Entscheidungstabellentechnik*, Heidelberg: I. H. Sauer-Verlag GmbH. - ISBN: 3793876187

[MOD55] Norm Def Stan 00-55:97, British Ministry of Defence (Hrsg.): *Requirements for safety related software in defence equipment*

[MTA98] MELLOR, Stephen J.; TOCKEY, Stephen R.; ARTHAUD, Rodolphe; LEBLANC, Phillipe: *An Action Language for UML: Proposal for a Precise Execution Semantics*; In: Lecture Notes In Computer Science, Vol. 1618, London: Springer, 1998. - ISBN 3540662529

[NAP04] NAUMANN, Peter; PACHL, Jörn: *Leit- und Sicherungstechnik im Bahnbetrieb - Fachlexikon*, 2. Aufl. Hamburg: Tetzlaff Verlag GmbH & Co. KG, 2004 - ISBN: 3878147023

[OMG06-1] OBJECT MANAGMENT GROUP (Hrsg.): *UML 2.0 OCL Specification*; OMG Dokument ptc/2005-06-06

[OMG06-2] OBJECT MANAGEMENT GROUP (Hrsg.): *Business Process Modeling Notation Specification*; OMG Dokument dtc/06- 01-01

[OMG07-1] OBJECT MANAGEMENT GROUP (Hrsg.): *Systems Modeling Language (SysML) Specification V1.0*; OMG Document formal/2007-09-01

[OMG07-2] OBJECT MANAGEMENT GROUP (Hrsg.): *Unified Modeling Language (UML): Superstructure, Version 2.1.1*; OMG Document formal/2007-02-05

[OMG07-3] OBJECT MANAGEMENT GROUP (Hrsg.): *Unified Modeling Language (UML): Infrastructure Definition, Version 2.1.2*; OMG Document formal/2007-11-04

[OMG07-4] OBJECT MANAGEMENT GROUP (Hrsg.): *MOF 2.0/XMI Mapping, Version 2.1.1;* OMG Document formal/2007-12-01

[OPR06] PLAN, Oliver; KALKBRENNER, A.; SANDERS, Ralf; LEMKE, Oliver; HUNGAR, Hardi: *OPRAIL Process Description*; Version 2.1; 2006

[OSC08-1] OSC EMBEDDED SYSTEMS AG (Hrsg.): *Rhapsody TestConductor*, http://www.osc-es.de/index.php?idcat=22, Abfruf vom 29.05.2008

[OSC08-2] OSC EMBEDDED SYSTEMS AG (Hrsg.): *Rhapsody ATG,* http://www.osc-es.de/index.php?idcat=23, Abruf vom 29.05.2008

[OSK06] OESTEREICH, Bernd, SCHRÖDER, Claudia; KLINK, Markus; ZOCKOLL, Guido: *OEP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte,* 1. Aufl. Heidelberg: Dpunkt Verlag, 2006. - ISBN: 3898644073

[PAC04] PACHL, Jörn: *Systemtechnik des Schienenverkehrs*, 4. Aufl. Wiesbaden: B. G. Teubner Verlag, 2004. - ISBN: 3519363836

[PET62] PETRI, Carl Adam: *Kommunikation mit Automaten*, Bonn: Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn, 1962

[ROB06] ROBERTSON, Suzanne; ROBERTSON, James: *Mastering the Requirements Process*, 2. Aufl. Amsterdam: Addison-Wesley Longman, 2006. - ISBN: 0321419499

[ROP99] ROPOHL, Günter: *Allgemeine Technologie - Eine Systemtheorie der Technik*, 2. Aufl. München: Hanser Fachbuchverlag, 1999. - ISBN 3446196064

[ROY70] ROYCE, Winston: *Managing the Development of Large Software Systems.* In: Proceedings of IEEE WESCON 26, August 1970, S. 1-9

[RQZ07] RUPP, Christian; ZENGLER, Barbara; QUEINS, Stefan: *UML 2 Glasklar*, 3. Aufl. München: Carl Hanser Verlag, 2007. - ISBN 3446411186

[RUP98] Rational Software Corporation (Hrsg.): *Rational Unified Process: Best Practices for Software Development Teams,* 1998

[SCH01] SCHNEIDER, Steve: *The B-method*. Palgrave Macmillan, 2001. - ISBN: 033379284X

[SCN99] SCHNIEDER, Eckehard: *Methoden der Automatisierung.* Braunschweig: Vieweg Verlagsgesellschaft, 1999. - ISBN: 3528065664

[STA98] STANDISH GROUP (Hrsg.): *A Recipe for Success,* 1998

[STM04] SCHINZ, Ingo; TOBEN, Tobe; MRUGALLA, Christian; WESPHAL, Bernd: *The Rhapsody UML Verification Environment,* S. 174-183, Second International Conference on Software Engineering and Formal Methods (SEFM'04), 2004

[STÖ04] STÖRRLE, Harald; Universität München, IFI: *Towards a formal Semantic of UML 2.0 Activities*; 2004.

[UPL06] UTTING, Mark; PRETSCHNER, Alexander; LEGEARD, Bruno; University of Waikato, Departement of Computer Sciences: *A Taxonomy of Model-based Testing*, 2006

[VIK05] VITOLINS, Valdis; KALNINS, Audris; University of Latvia: *Sematics of UML 2.0 Acitvity Diagrams for Business Modeling by Means of Virtual Machine*; 2005

[WEI06] WEILKIENS, Tim: *Systems Engineering mit SysML/UML*, 1. Aufl. Heidelberg: Dpunkt Verlag, 2006. - ISBN 389864409X

[WES00] WEISSTEIN, Eric W.: *Deterministic*.
In: http://wolfram.mathworld.com/Deterministic.html, Abruf vom 10.02.2008

A. Metamodell-Blockdiagramme des funktionalen Modells

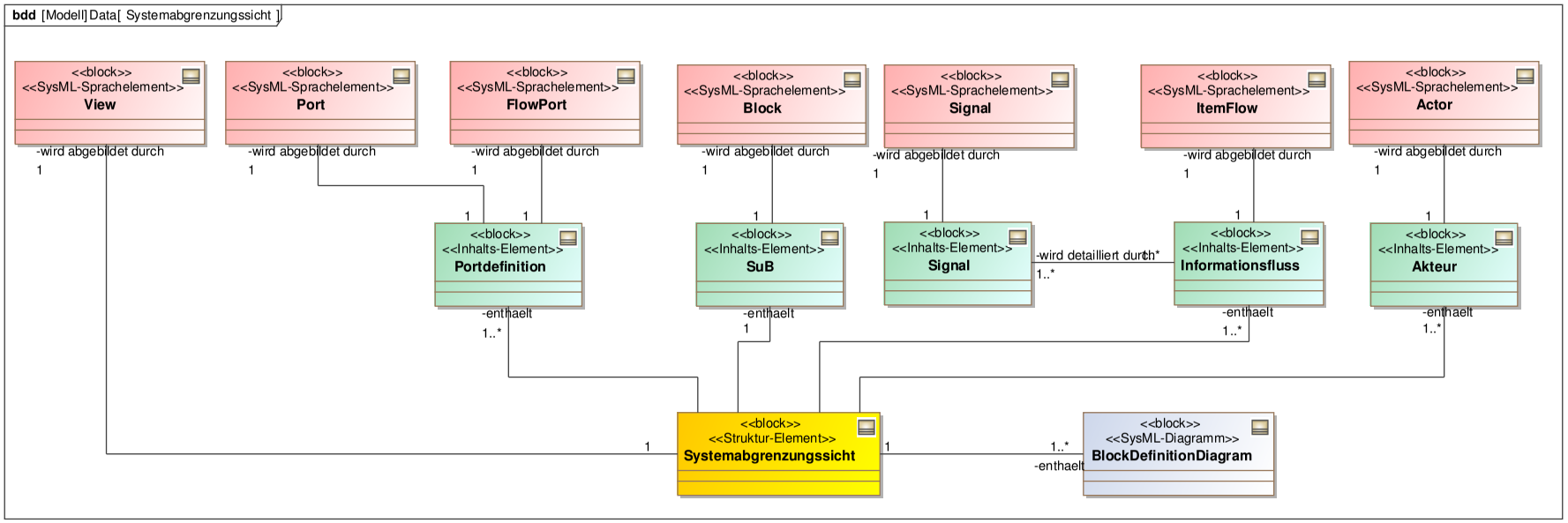


Abbildung A.1.: Metmodelldefinition der Systemabgrenzungssicht

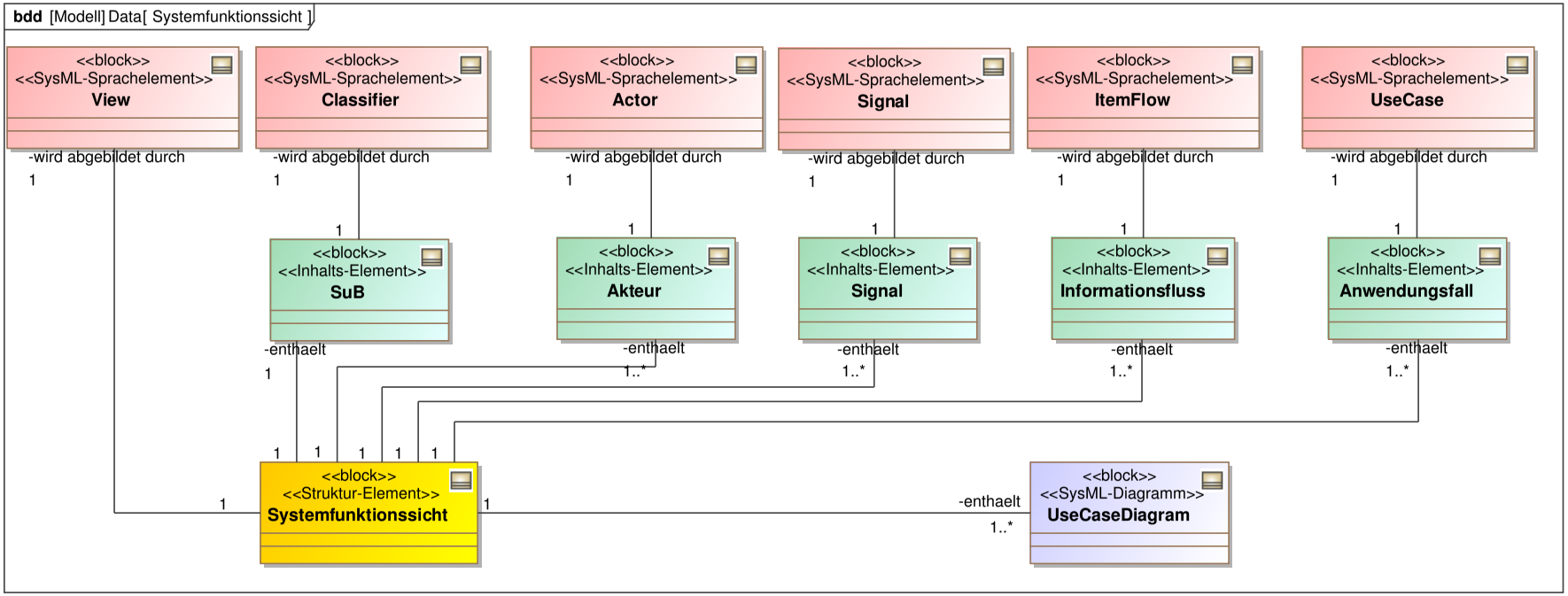


Abbildung A.2.: Metamodelldefinition der Systemfunktionssicht

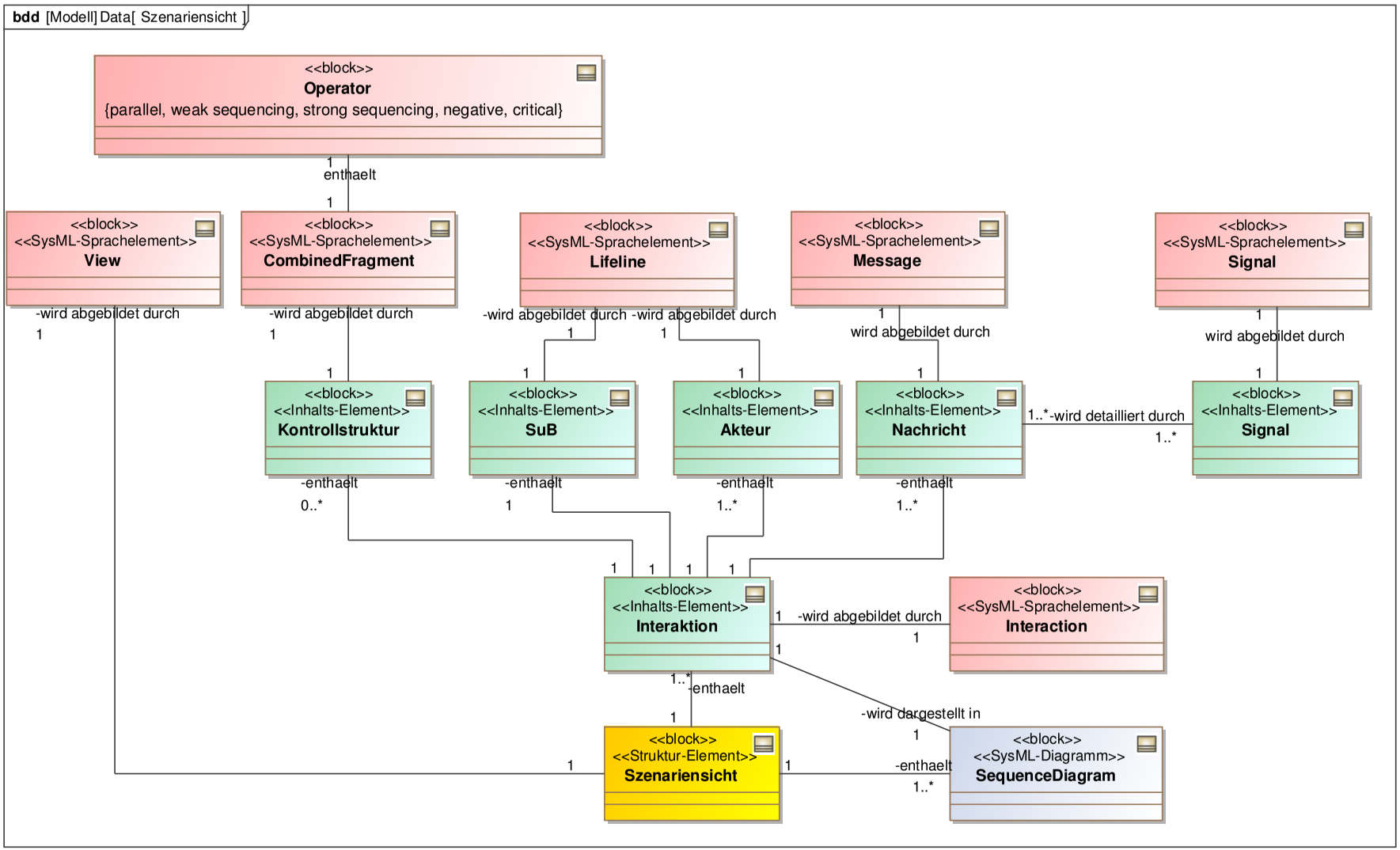


Abbildung A.3.: Metamodell der Szenariensicht

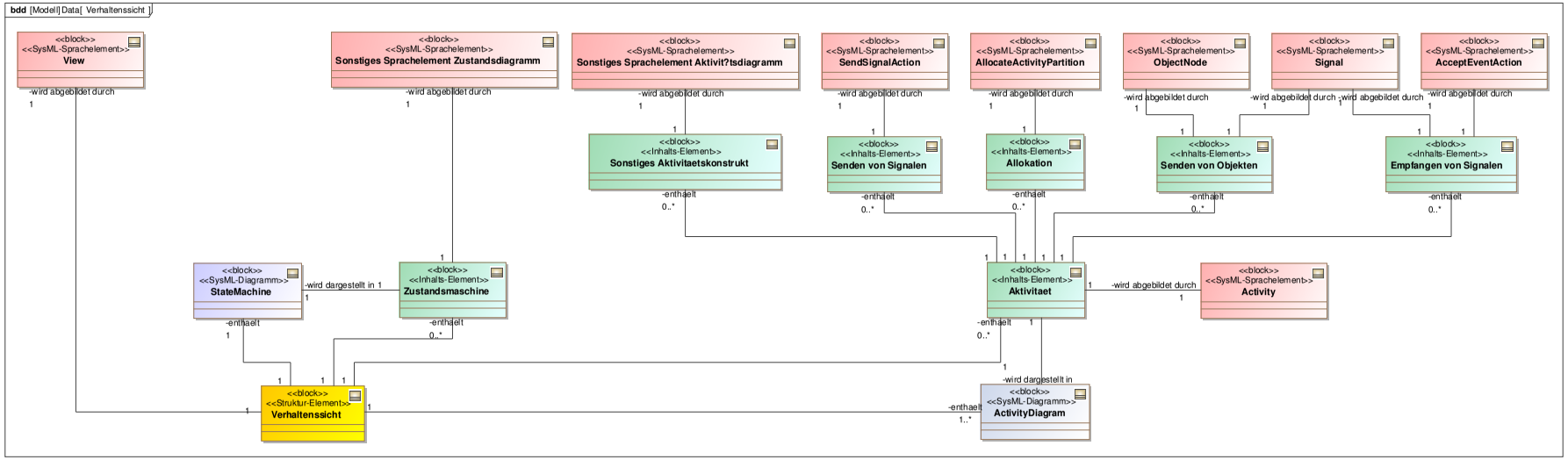


Abbildung A.4.: Metamodell der Systemverhaltenssicht

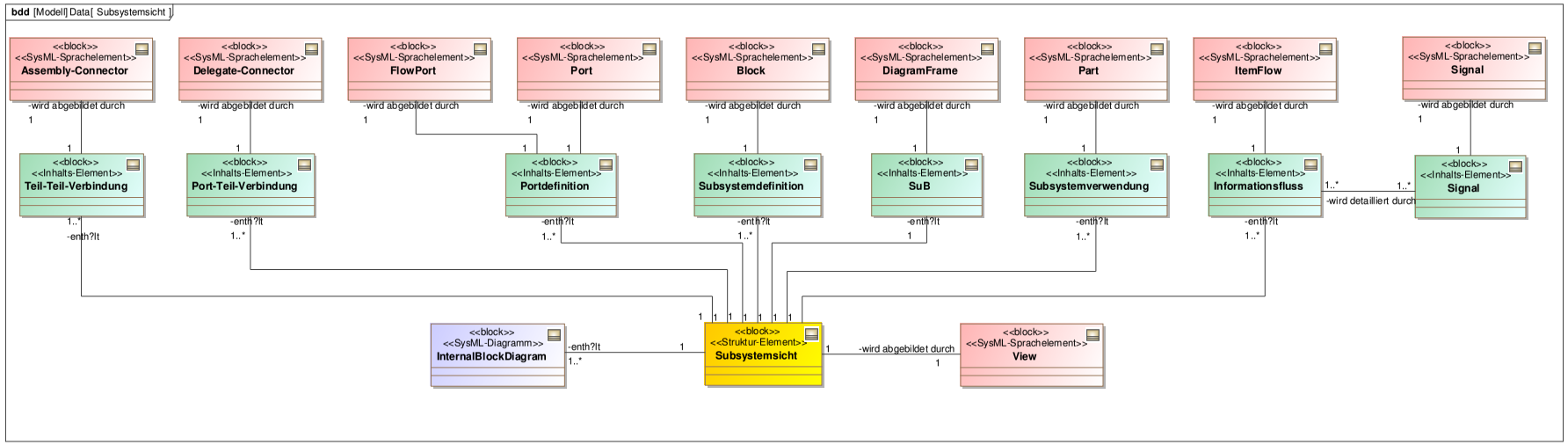


Abbildung A.5.: Metamodell der Subsystemsicht

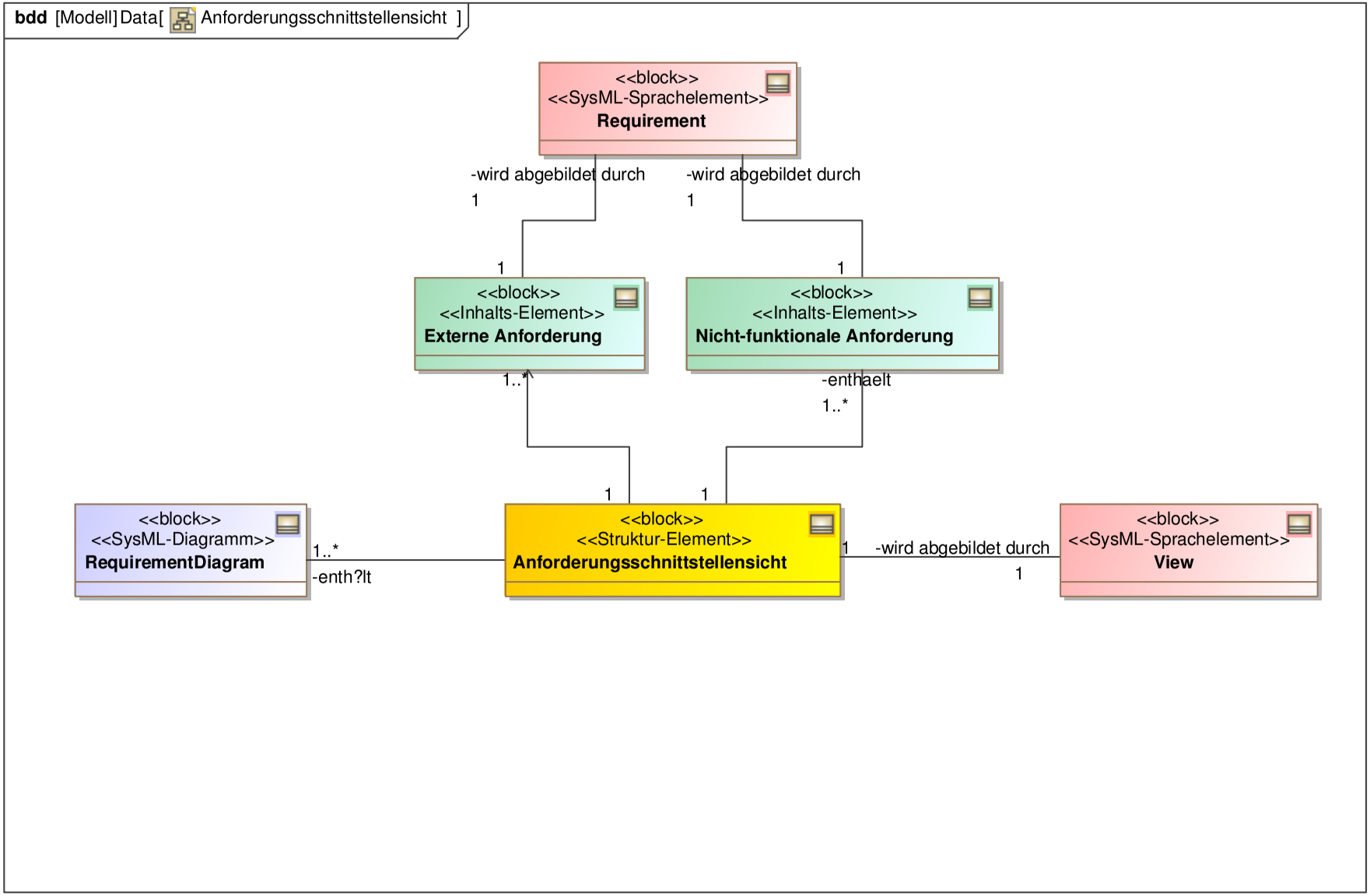


Abbildung A.6.: Metamodell der Anforderungsschnittstellensicht

B. Metamodell-Aktivitätsdiagramme des Prozessmodells

B.1. Phasen

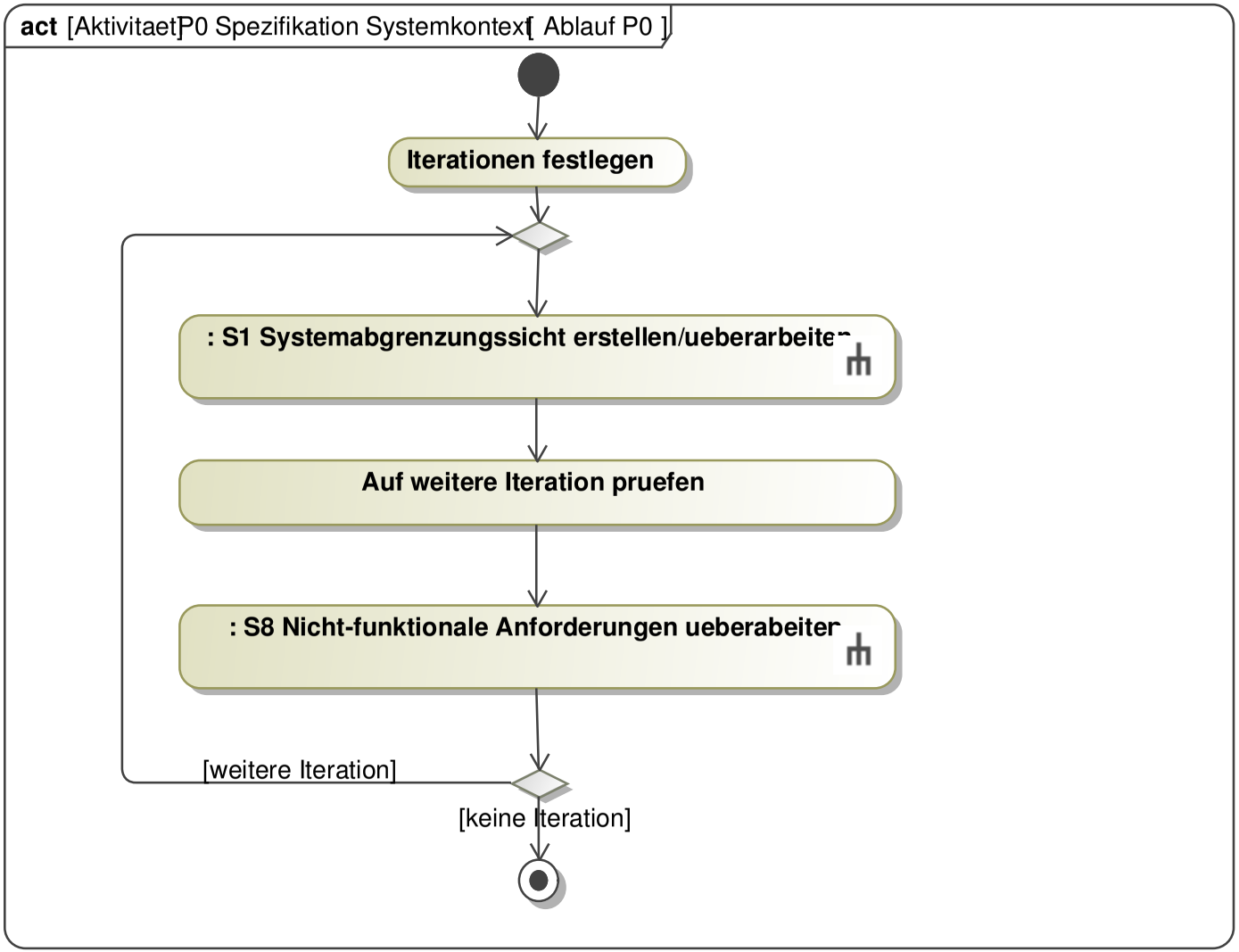


Abbildung B.1.: Aktivitätsdiagramm Phase P0

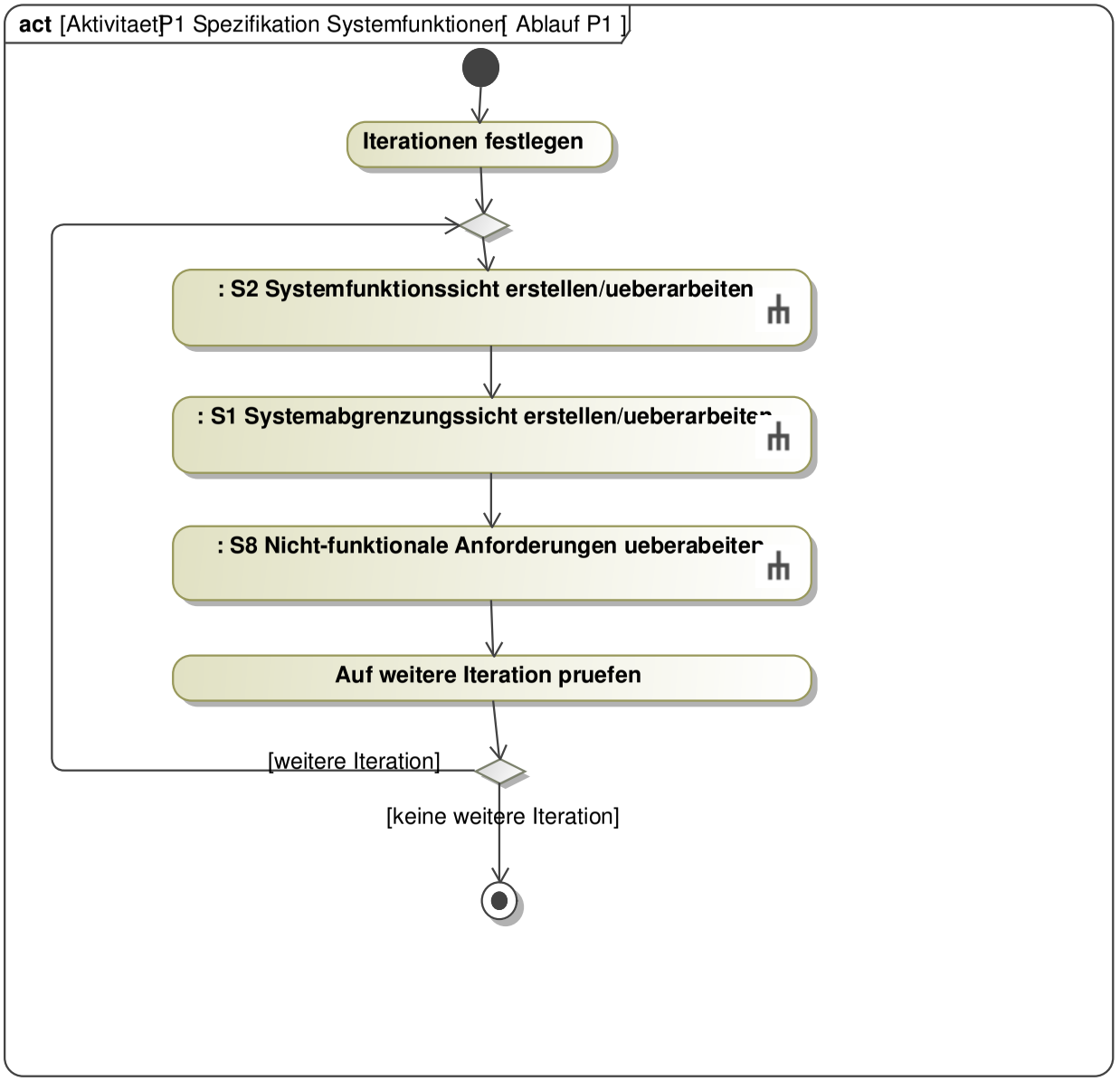


Abbildung B.2.: Aktivitätsdiagramm Phase P1

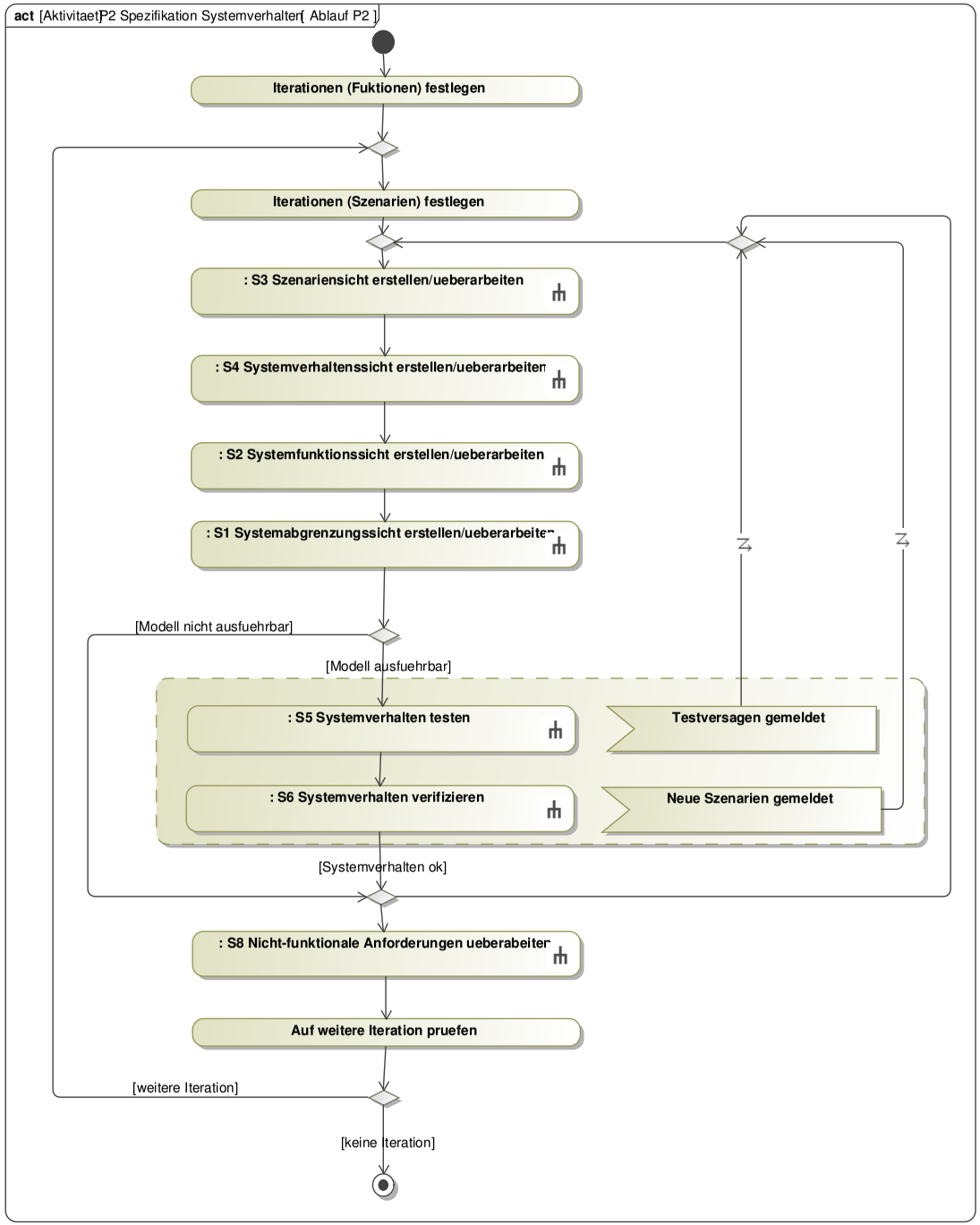


Abbildung B.3.: Aktivitätsdiagramm Phase P2

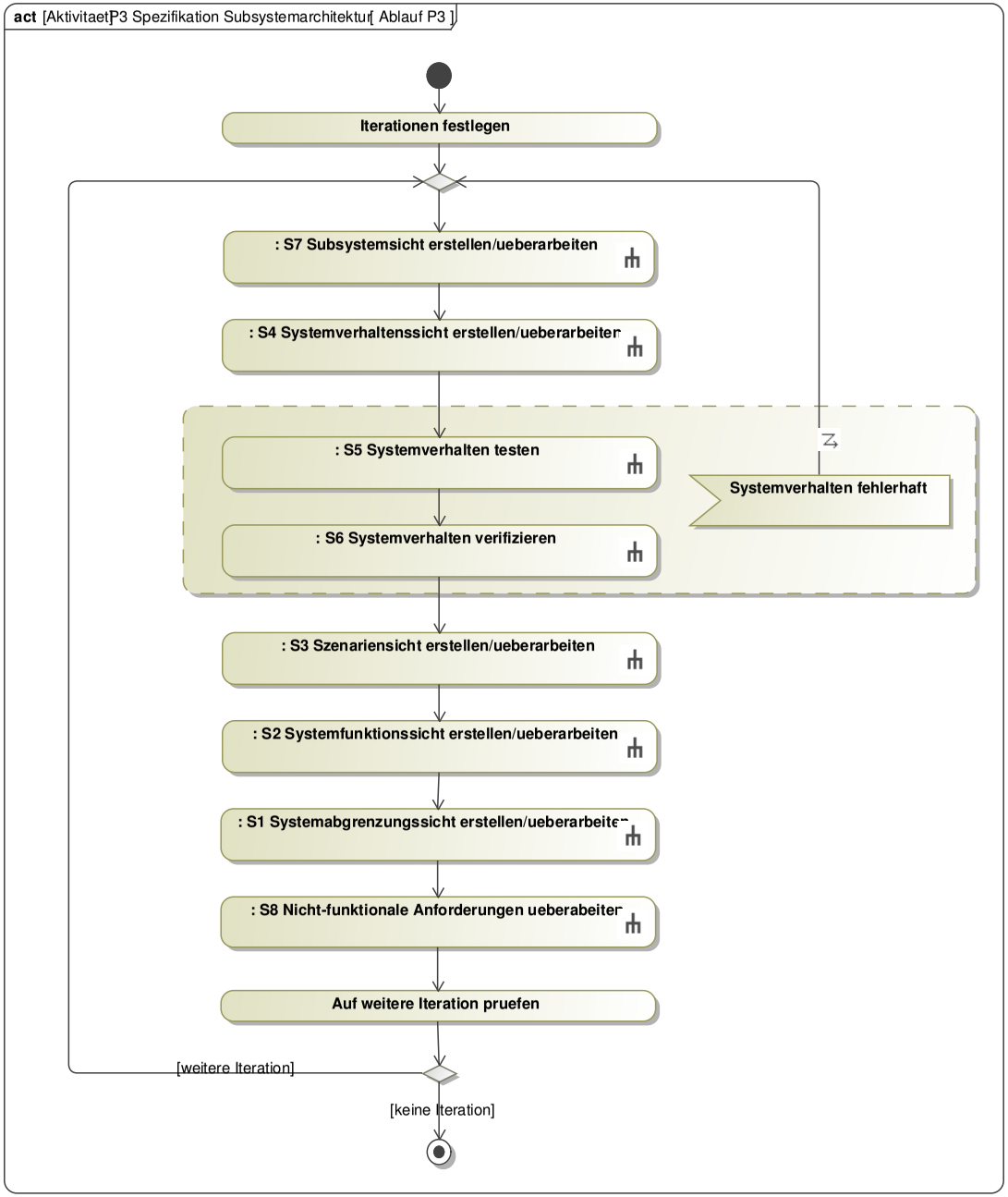


Abbildung B.4.: Aktivitätsdiagramm Phase P3

B.2. Subprozesse

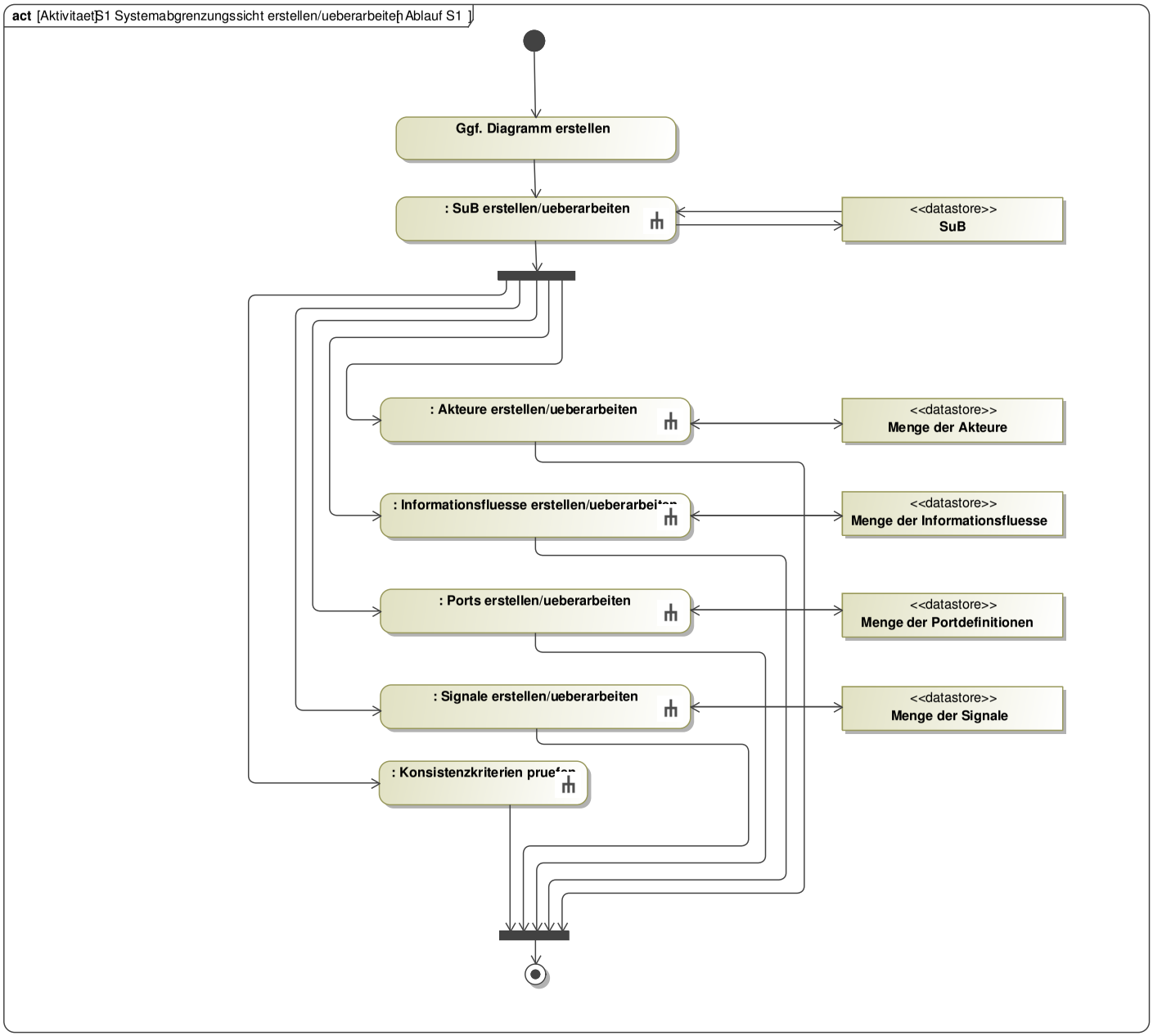


Abbildung B.5.: Aktivitätsdiagramm Subprozess S1

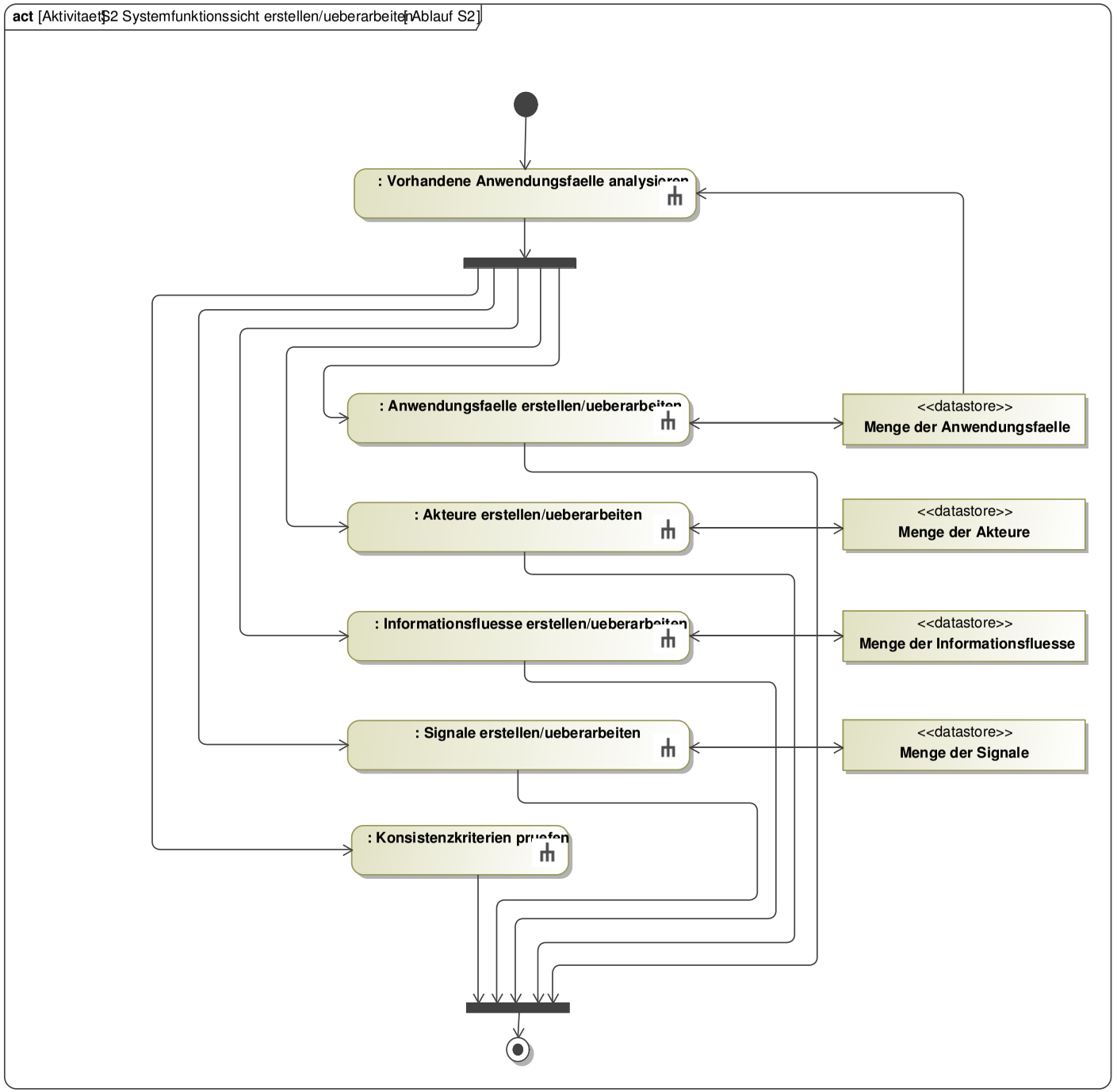


Abbildung B.6.: Aktivitätsdiagramm Subprozess S2

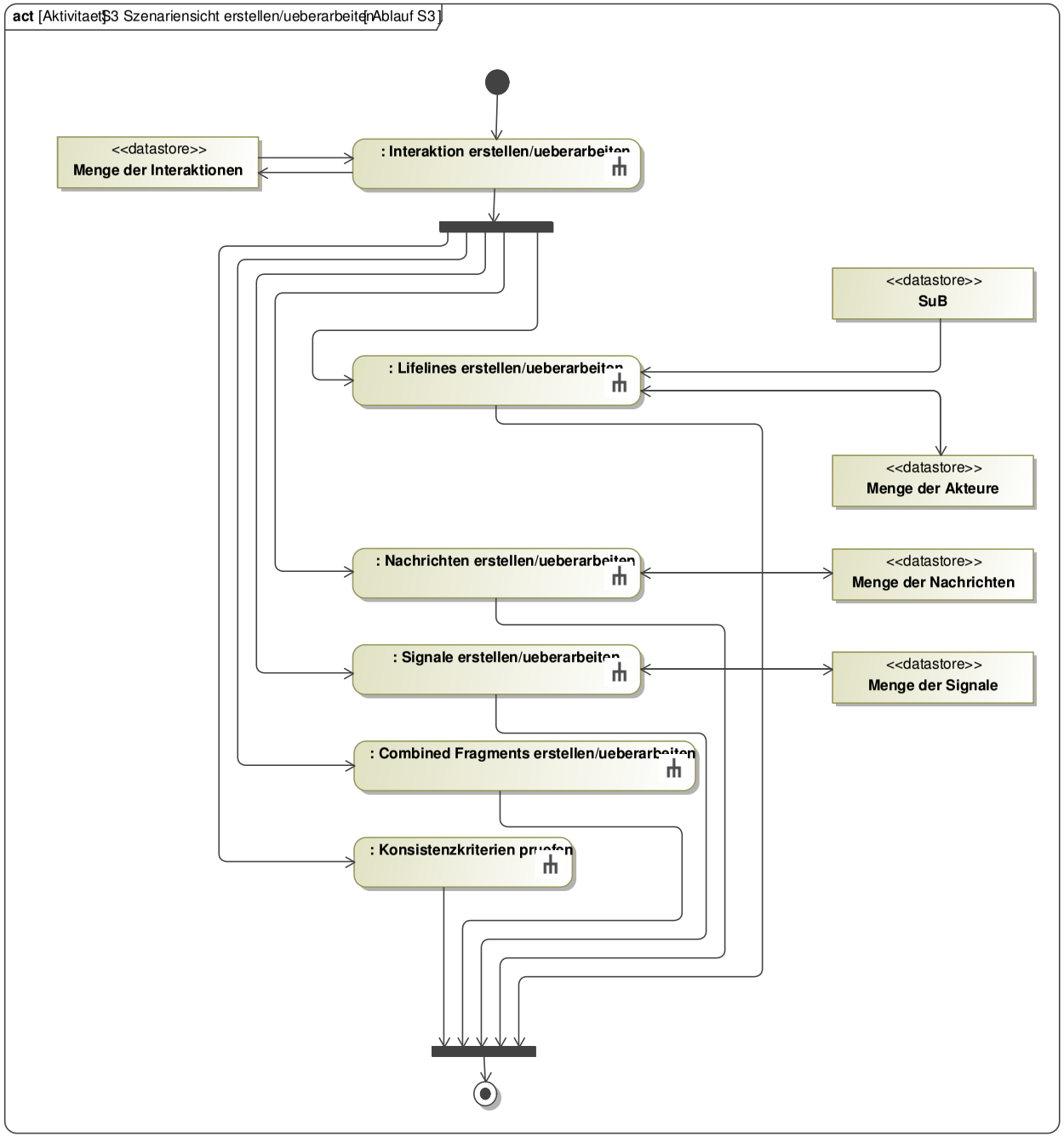


Abbildung B.7.: Aktivitätsdiagramm Subprozess S3

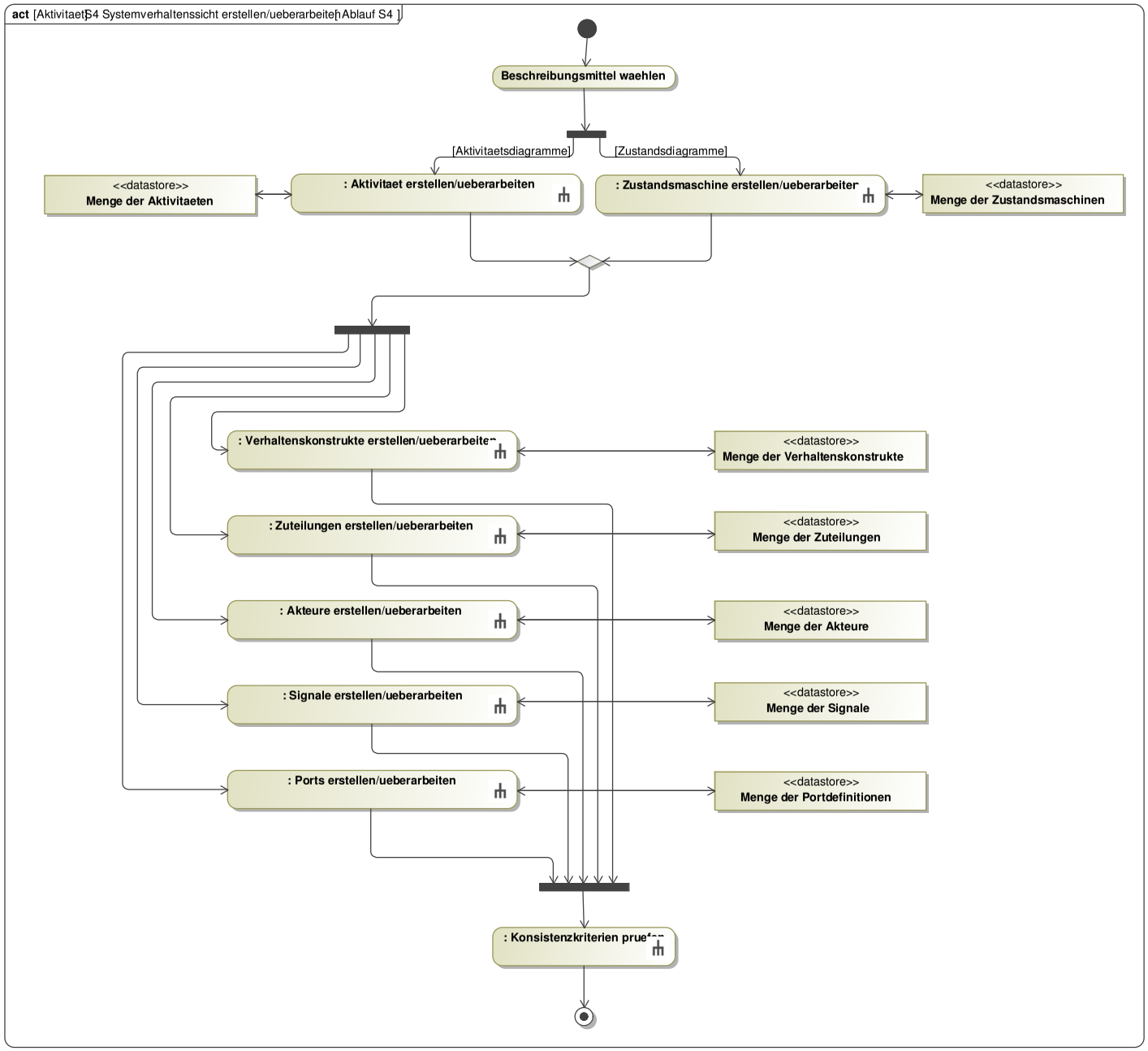


Abbildung B.8.: Aktivitätsdiagramm Subprozess S4

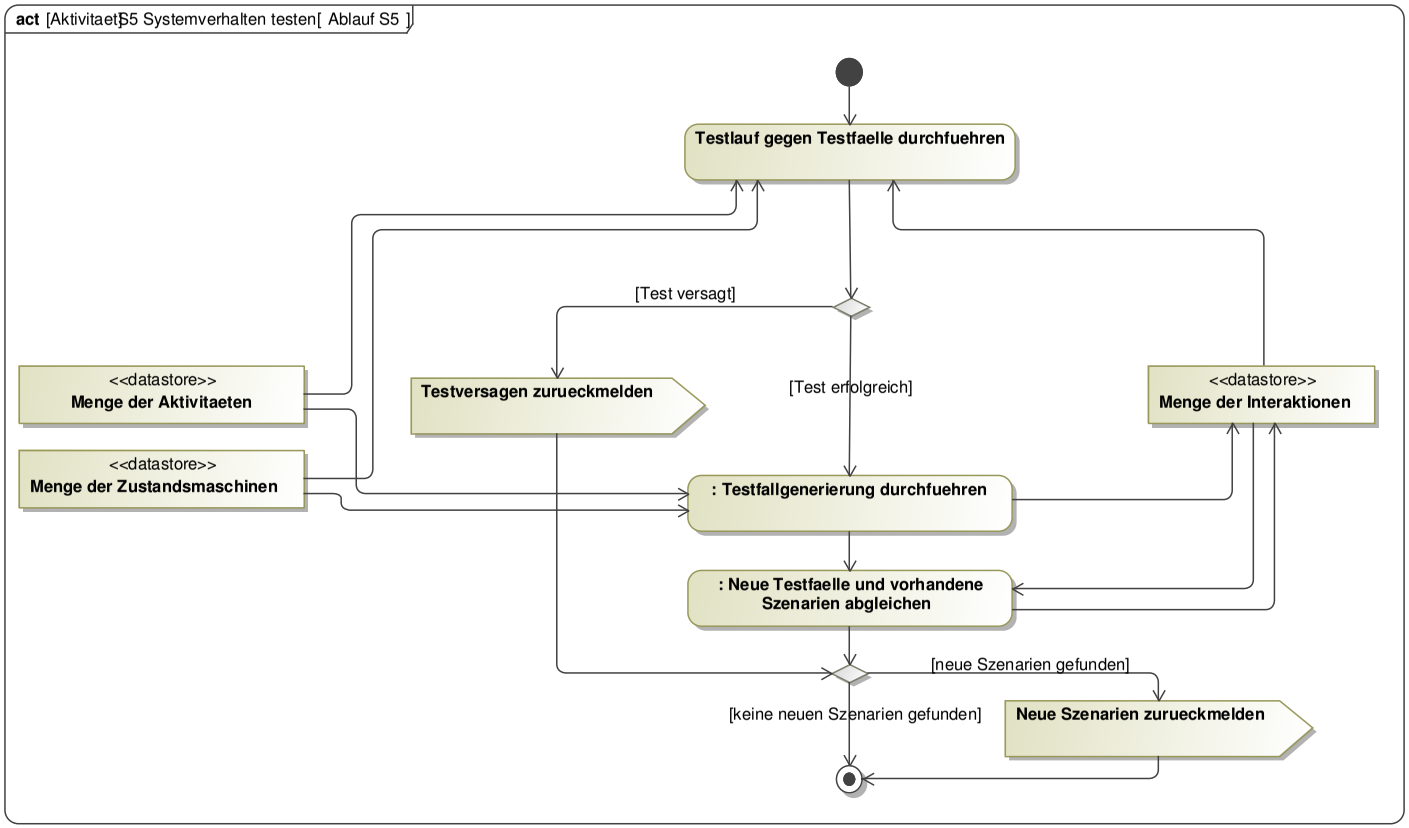


Abbildung B.9.: Aktivitätsdiagramm Subprozess S5

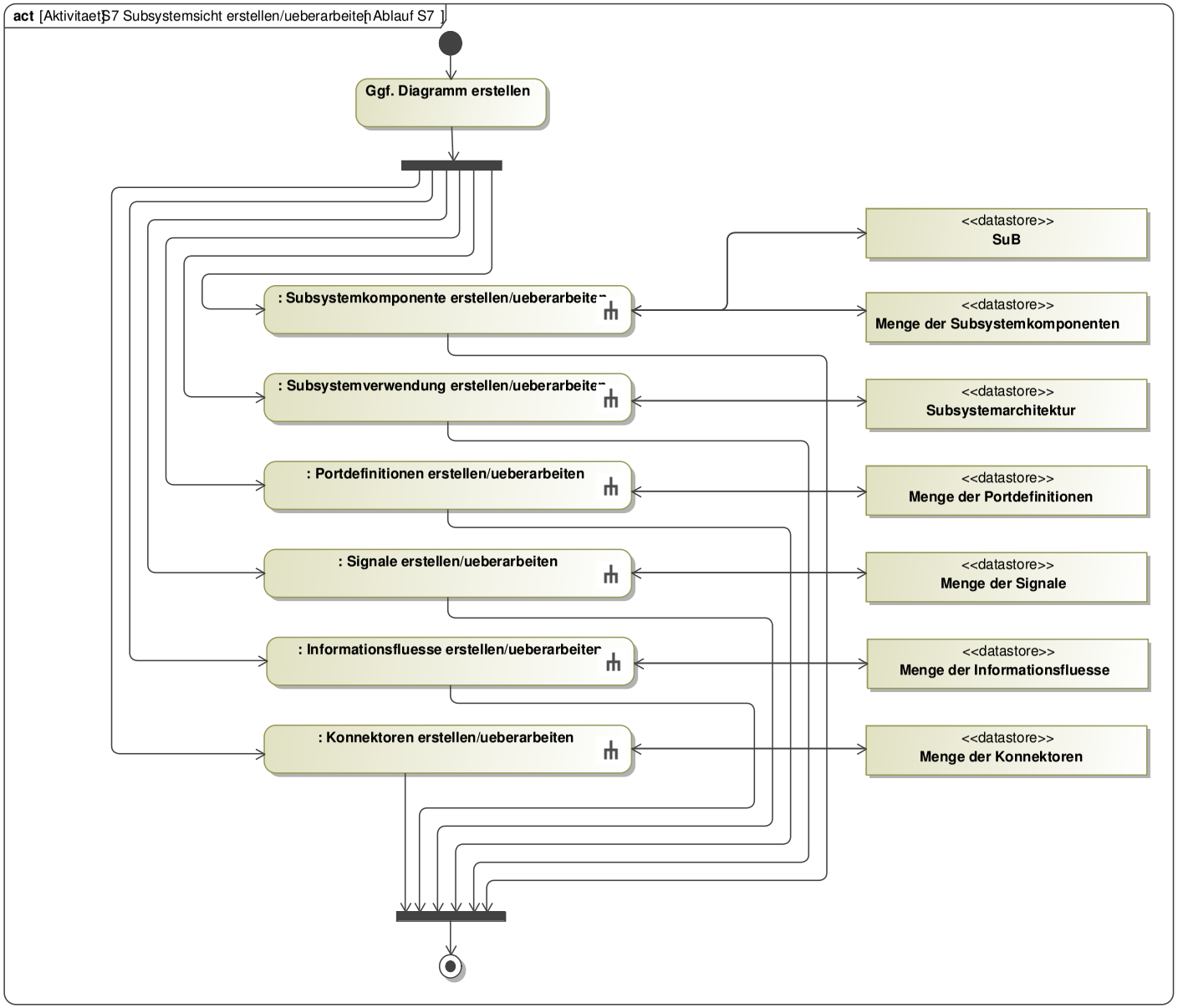


Abbildung B.10.: Aktivitätsdiagramm Subprozess S7

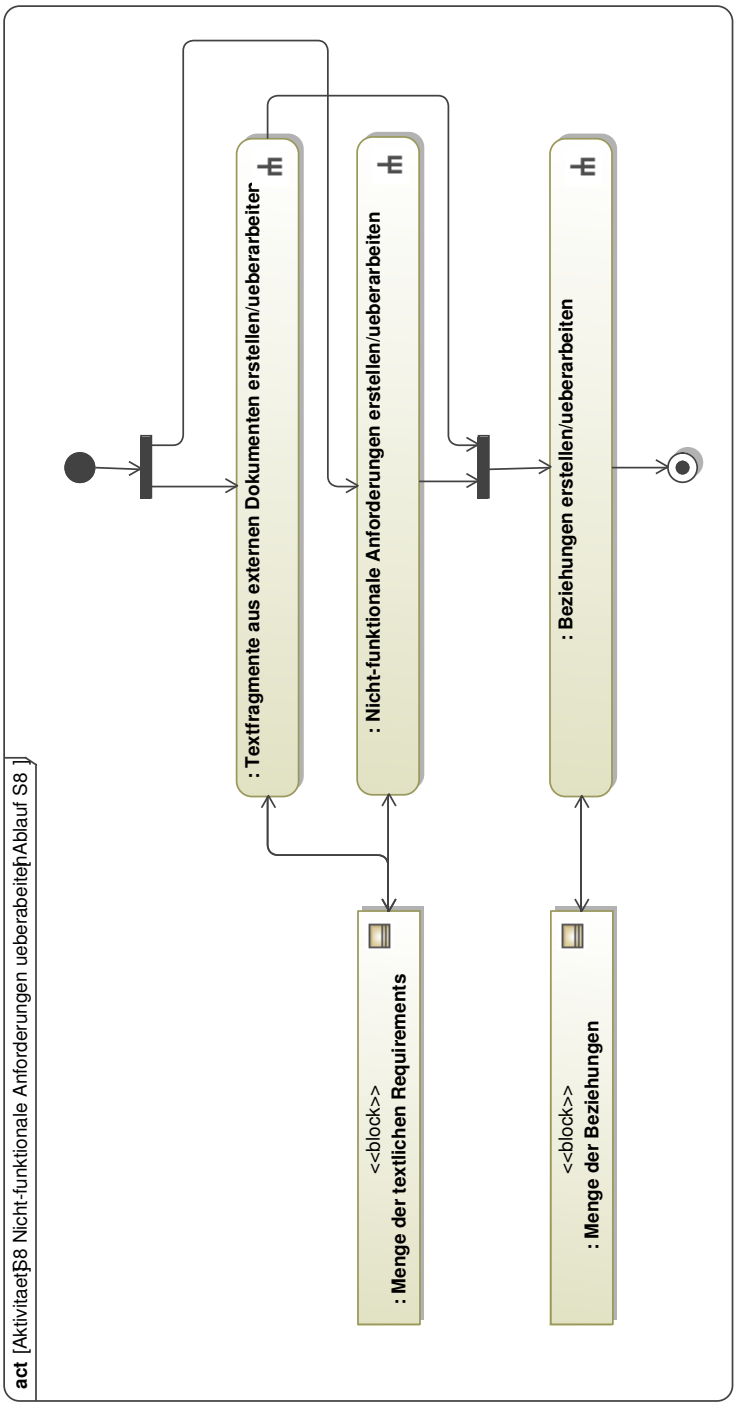


Abbildung B.11.: Aktivitätsdiagramm Subprozess S8